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

[← Home](https://imageslr.com/)

**📝【LeetCode】一个模板通杀所有「二分查找」问题**

Mar 15, 2020 • 1140 View

* [引言](https://imageslr.com/2020/03/15/binary-search.html#%E5%BC%95%E8%A8%80)
* [找下界](https://imageslr.com/2020/03/15/binary-search.html#%E6%89%BE%E4%B8%8B%E7%95%8C)
  + [问题定义](https://imageslr.com/2020/03/15/binary-search.html#%E9%97%AE%E9%A2%98%E5%AE%9A%E4%B9%89)
  + [思路描述](https://imageslr.com/2020/03/15/binary-search.html#%E6%80%9D%E8%B7%AF%E6%8F%8F%E8%BF%B0)
  + [模板代码](https://imageslr.com/2020/03/15/binary-search.html#%E6%A8%A1%E6%9D%BF%E4%BB%A3%E7%A0%81)
* [找上界](https://imageslr.com/2020/03/15/binary-search.html#%E6%89%BE%E4%B8%8A%E7%95%8C)
* [查找指定值第一次出现的位置](https://imageslr.com/2020/03/15/binary-search.html#%E6%9F%A5%E6%89%BE%E6%8C%87%E5%AE%9A%E5%80%BC%E7%AC%AC%E4%B8%80%E6%AC%A1%E5%87%BA%E7%8E%B0%E7%9A%84%E4%BD%8D%E7%BD%AE)
* [查找指定值最后一次出现的位置](https://imageslr.com/2020/03/15/binary-search.html#%E6%9F%A5%E6%89%BE%E6%8C%87%E5%AE%9A%E5%80%BC%E6%9C%80%E5%90%8E%E4%B8%80%E6%AC%A1%E5%87%BA%E7%8E%B0%E7%9A%84%E4%BD%8D%E7%BD%AE)
* [查找指定值的位置](https://imageslr.com/2020/03/15/binary-search.html#%E6%9F%A5%E6%89%BE%E6%8C%87%E5%AE%9A%E5%80%BC%E7%9A%84%E4%BD%8D%E7%BD%AE)
* [总结：模板代码](https://imageslr.com/2020/03/15/binary-search.html#%E6%80%BB%E7%BB%93%E6%A8%A1%E6%9D%BF%E4%BB%A3%E7%A0%81)
* [对比：左闭右开的写法](https://imageslr.com/2020/03/15/binary-search.html#%E5%AF%B9%E6%AF%94%E5%B7%A6%E9%97%AD%E5%8F%B3%E5%BC%80%E7%9A%84%E5%86%99%E6%B3%95)
  + [两者对比](https://imageslr.com/2020/03/15/binary-search.html#%E4%B8%A4%E8%80%85%E5%AF%B9%E6%AF%94)
  + [模板代码](https://imageslr.com/2020/03/15/binary-search.html#%E6%A8%A1%E6%9D%BF%E4%BB%A3%E7%A0%81-1)
  + [补充说明](https://imageslr.com/2020/03/15/binary-search.html#%E8%A1%A5%E5%85%85%E8%AF%B4%E6%98%8E)
* [总结](https://imageslr.com/2020/03/15/binary-search.html#%E6%80%BB%E7%BB%93)

本文涉及到的 LeetCode 题目：

* [LeetCode 34. 在排序数组中查找元素的第一个和最后一个位置](https://leetcode-cn.com/problems/find-first-and-last-position-of-element-in-sorted-array/)
* [LeetCode 35. 搜索插入位置](https://leetcode-cn.com/problems/search-insert-position/)

个人笔记

引言

二分查找有很多应用场景。可以说，**只要问题对应的函数图像在给定区间是单调的，那就可以使用二分查找在这个区间搜索目标值**。

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。

找下界

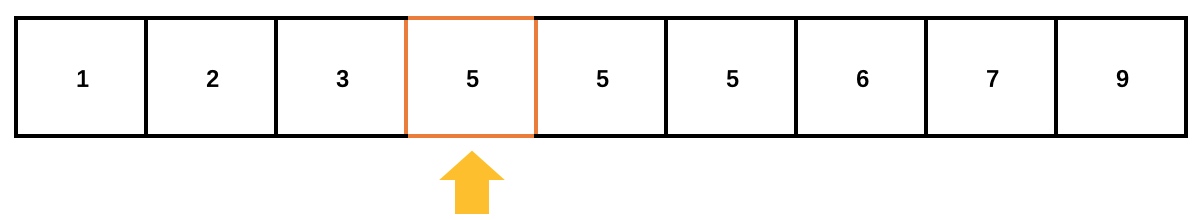
问题定义

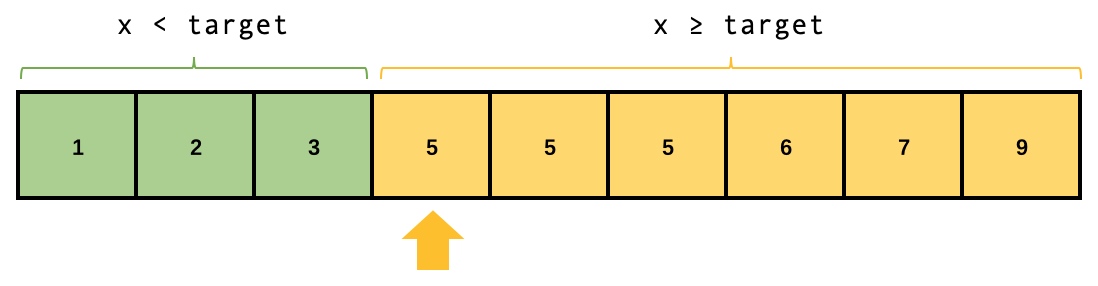
给定一个升序排列的数组，我们将满足 x ≥ target 的**第一个元素**定义为「下界」。给定一个目标值 target，要求返回其下界的下标。如果下界不存在，返回数组长度。

比如：对于数组 [0,1,2,3,4]，当 target=3 时，返回下标 3；当 target=5 时，返回下标 5。

C++ STL 中的 lower\_bound() 函数就实现了这个功能。

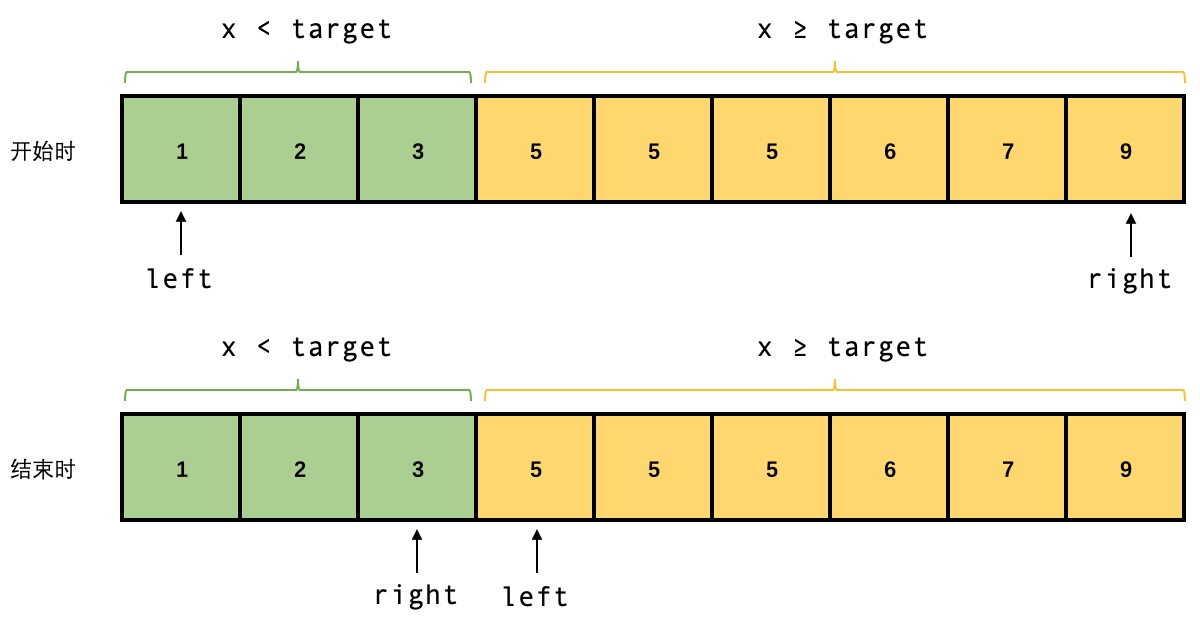
思路描述

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：  


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**：  


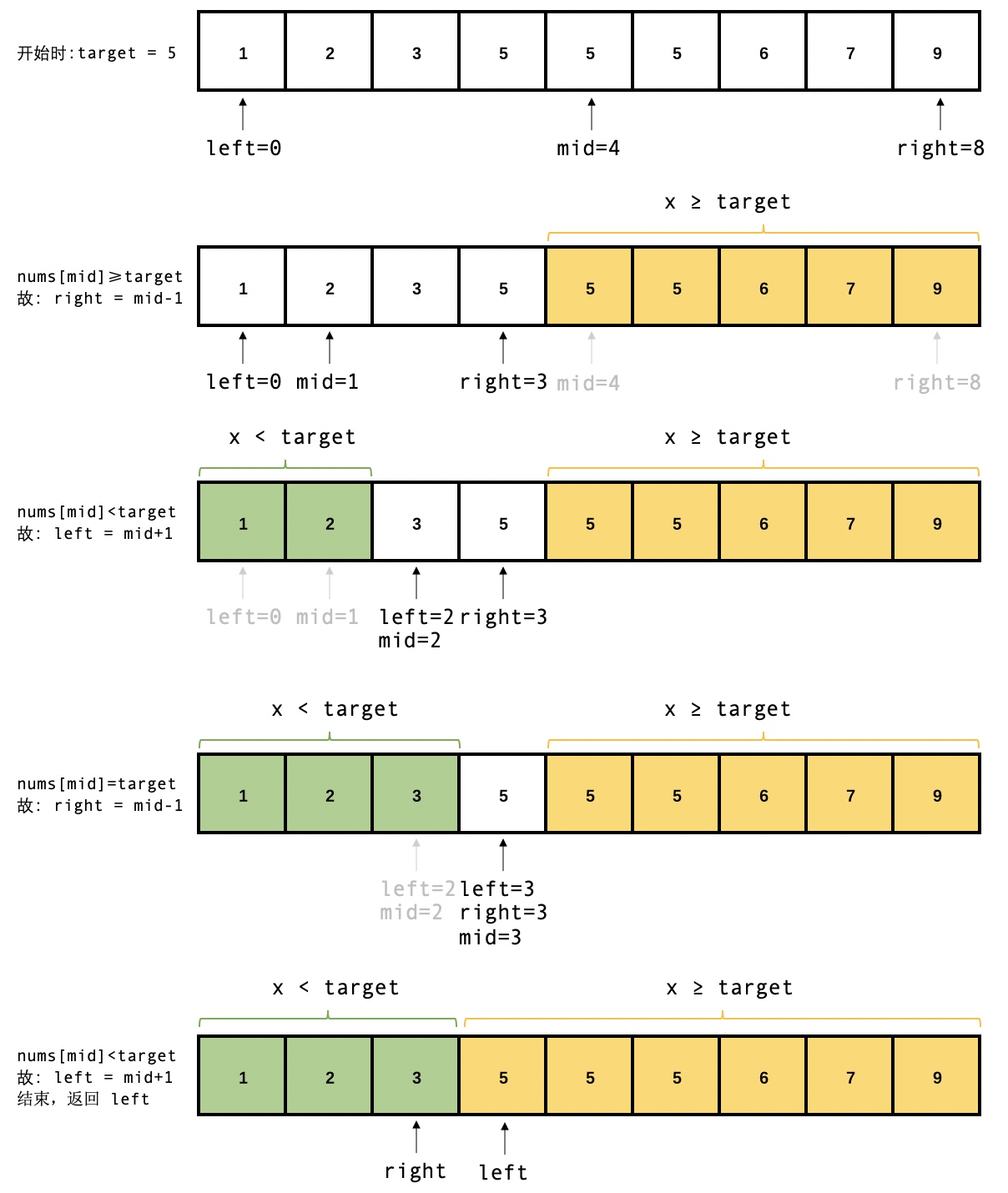
接下来，我们使用「闭区间」的写法来描述思路。先定义几个变量：

* 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**：  


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：  


模板代码

// 查找满足 x ≥ target 的下界的下标

func LowerBound(nums []int, target int) int {

left, right := 0, len(nums)-1

for left <= right {

mid := left + (right-left) >> 1

if nums[mid] >= target { // 这里的比较运算符与题目要求一致

right = mid - 1

} else {

left = mid + 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。「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**。

最后注意一些细节：

* left、right 的初值为 0、n-1，表示「闭区间」
* 循环的判定条件是 left <= right，表示区间不为空
* 更新 left 和 right 时均跳过了中间元素 mid

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。接下来，我们看看如何使用这一个模板，通杀所有二分查找问题。

找上界

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。比如：对于数组 [0,1,2,3,4]，当 target=3 时，返回下标 2；当 target=5 时，返回下标 4。

根据上界和下界的定义，我们可以发现：**上界和「互补的」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：

*// 查找满足 x > target 的下界的下标*

func LowerBound(nums []int, target int) int {

*// ...*

**if** nums[mid] > target { *// 只需将这里改为 >*

*// ...*

}

然后将下界的下标减一，就是我们要找的上界：

*// 查找满足 x ≤ target 的上界的下标*

func UpperBound(nums []int, target int) int {

return LowerBound(nums, target)-1

}

或者，我们可以直接将 LowerBound 代码中的返回语句改为 left-1 或者 right，就能得到一个纯净的、无依赖的 UpperBound：

func UpperBound(nums []**int**, **target** **int**) **int** {

*// ...*

**if** nums[mid] > **target** { *// 这里是 >*

*// ...*

**return** right *// 或者返回 left-1*

}

可以看到，找下界的模板代码略作修改，就能用来查找上界了！

查找指定值第一次出现的位置

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。只需在模板代码中增加一个判断：

func searchFirst(nums []int, target int) int {

left, right := 0, len(nums)-1

for left <= right {

mid := left + (right-left) >> 1

if nums[mid] >= target {

right = mid - 1

} else {

left = mid + 1

}

}

+ if left >= len(nums) || nums[left] != target { // 判断一下是否越界，或者不相等

+ return -1

+ }

return left

}

查找指定值最后一次出现的位置

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，直接调用模板代码：

func searchLast(nums []int, target int) int {

left, right := 0, len(nums)-1

for left <= right {

mid := left + (right-left) >> 1

if nums[mid] > target { // 这里是 > 而不是 >=

right = mid - 1

} else {

left = mid + 1

}

}

if right < 0 || nums[right] != target { // 判断一下是否越界，或者不相等

return -1

}

return right // 这里返回 right 而不是 left

}

这两道题对应于 [LeetCode 34. 在排序数组中查找元素的第一个和最后一个位置](https://leetcode-cn.com/problems/find-first-and-last-position-of-element-in-sorted-array/)。有了上面两个函数，题解代码仅需一行：

func searchRange(nums []int, target int) []int {

return []int{searchFirst(nums, target), searchLast(nums, target)}

}

查找指定值的位置

这是最基本的二分查找问题，对应于 [LeetCode 35. 搜索插入位置](https://leetcode-cn.com/problems/search-insert-position/)：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：

func searchInsert(nums []int, target int) int {

left, right := 0, len(nums)-1

**for** left <= right {

mid := left+(right-left)>>1

**if** nums[mid] >= target {

right = mid-1

} **else** {

left = mid+1

}

}

return left

}

target 按顺序插入的位置，就是满足 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。代码略。

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

对比：左闭右开的写法

两者对比

上面我们采用了「闭区间」的写法，这种情况下：

1. 区间范围是 [left,right]
2. 循环条件是「小于等于」，表示 [left,right] 不为空
3. right 的初值为「最大值」
4. left、right 分别需要「±1」，才能使新区间不包含 mid
5. 区间为空时，left 指向下界，right 指向*互补条件*的上界
6. 如果需要下界，只能返回 left

另一种常见的写法是「左闭右开」，比如 C++ 标准库 <algorithm> 中就采用了这种写法，带来的变化是：

1. 区间范围是 [left, right)
2. 循环条件变为「小于」，表示 [left,right) 不为空
3. right 的初值为「最大值+1」
4. right 直接赋值为 mid，不需要 -1 就能使新区间不包含 mid
5. 区间为空时，left、right 都指向下界（它们重合）
6. 如果需要下界，可以返回任意一个！

模板代码

func LowerBound(nums []int, target int) int {

left, right := 0, len(nums) // 3. 下标的最大值为 n-1，故 right 初值为 n，即「最大值+1」

for left < right { // 2. 循环条件为「小于」

mid := left + (right-left) >> 1

if nums[mid] >= target {

right = mid // 4. right 不需要 -1

} else {

left = mid + 1

}

}

return left // 6. 返回 left、right 均可以

}

补充说明

有人认为「左闭右开」这种写法的优点是：

1. 当区间为空时，「左闭右开」是 left == right，而「闭区间」是 left > right，前者更为直观。比如：0 ≤ a < 0 和 0 ≤ a ≤ -1，前者更符合人类直觉
2. 另外在这种情况下，返回 left 和 right 均可，因为它们重合。而「闭区间」只能返回 first

但在我看来，无论哪种写法，只要理解了思路，就都能很容易地将它们写出来。至于更喜欢哪种写法，就见仁见智了。

总结

本文主要介绍了一种通用的二分查找下界的模板代码，理解其原理后，不需要背模板，也可以自然地将代码写出来。

本文得出了以下结论：

* 二分查找无论是找下界、还是找上界、还是找特定值，都可以套用同一个模板代码
* 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
* 「左闭右开」和「闭区间」的写法本质上都是相同的原理，只要理解了本文的内容，选择哪种写法都没有问题

在后续的文章中，我们将继续使用这个模板，解决更多的实际问题，请阅读：[在实际问题中运用二分查找模板代码](https://imageslr.com/2020/03/15/leetcode-875.html)。

<https://imageslr.com/2020/03/15/leetcode-875.html>

本文发表在我的博客 <https://imageslr.com/>。我也会分享更多的题解，一起交流，共同进步！

[← Home](https://imageslr.com/)

# 📝【LeetCode】在实际问题中运用二分查找模板代码

Mar 15, 2020 • 202 View

* [引言](https://imageslr.com/2020/03/15/leetcode-875.html#%E5%BC%95%E8%A8%80)
* [875. 爱吃香蕉的珂珂](https://imageslr.com/2020/03/15/leetcode-875.html#875-%E7%88%B1%E5%90%83%E9%A6%99%E8%95%89%E7%9A%84%E7%8F%82%E7%8F%82)
* [1011. 在 D 天内送达包裹的能力](https://imageslr.com/2020/03/15/leetcode-875.html#1011-%E5%9C%A8-d-%E5%A4%A9%E5%86%85%E9%80%81%E8%BE%BE%E5%8C%85%E8%A3%B9%E7%9A%84%E8%83%BD%E5%8A%9B)
* [总结](https://imageslr.com/2020/03/15/leetcode-875.html#%E6%80%BB%E7%BB%93)

## 引言

在上一篇文章中，我们介绍了「二分查找」的通用模板，并通过几个例题说明了如何套用这个模板。建议读者先阅读上一篇文章：[一个模板通杀所有「二分查找」问题](https://imageslr.com/2020/03/15/binary-search.html)。

简要概括上一篇文章的主要内容：

* 二分查找适用于所有**「在单调区间中搜索目标值」**的问题
* 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（下界）、查找最后一个小于等于特定值的元素（上界）… 这些问题都可以通过套用同一个**「查找下界」的模板**来解决：

// 查找满足 x ≥ target 的下界的下标

func LowerBound(nums []int, target int) int {

left, right := 0, len(nums)-1

for left <= right {

mid := left + (right-left) >> 1

if nums[mid] >= target { // 这里的比较运算符与题目要求一致

right = mid - 1

} else {

left = mid + 1

}

}

return left // 返回下界的下标

}

本文将通过两个实际问题，深入讲解如何运用这个模板。

## 875. 爱吃香蕉的珂珂

这道题是 [LeetCode 875 题](https://leetcode-cn.com/problems/koko-eating-bananas/)：

珂珂喜欢吃香蕉。这里有 N 堆香蕉，第 i 堆中有 piles[i] 根香蕉。警卫已经离开了，将在 H 小时后回来。

珂珂可以决定她吃香蕉的速度 K （单位：根/小时）。每个小时，她将会选择一堆香蕉，从中吃掉 K 根。如果这堆香蕉少于 K 根，她将吃掉这堆的所有香蕉，然后这一小时内不会再吃更多的香蕉。

珂珂喜欢慢慢吃，但仍然想在警卫回来前吃掉所有的香蕉。

返回她可以**在 H 小时内吃掉所有香蕉的最小速度 K**（K 为整数）。

首先分析题目：珂珂每小时只能选择某一堆香蕉，吃掉其中的 K 根；如果这一堆香蕉不够 K 根，那么珂珂吃完之后，也必须等到下一个小时才能继续吃另一堆。要求返回她可以在 H 小时内吃掉所有香蕉的**最小速度** K。

显然，一种比较简单的方式是从 1 开始，依次递增 1，遍历所有可能的速度，返回第一个可以在 H 小时内吃掉所有香蕉的速度。

不过观察本题，我们容易发现：速度越快，吃掉所有香蕉的时间就越短。也就是说，**搜索区间是单调递减的**，因此可以使用二分查找。另外，本题要找的是可以在 H 小时内吃掉所有香蕉的**最小速度**，这实际上就是要查找**下界**。所以可以直接套用上面的模板代码：

**func** **minEatingSpeed**(piles []**int**, H **int**) **int** {

*// left, right，mid 的含义是「吃香蕉的速度」*

*// 每小时最少吃一根香蕉，最多只能吃一堆香蕉，所以 left、right 的初值分别为 1、MaxOf(piles)*

left, right := 1, MaxOf(piles)

**for** left <= right {

mid := left + (right-left)>>1

*// 假设在 H 小时内「恰好」吃掉所有香蕉的速度为 targetSpeed，则判断条件可以写为：*

*// if mid >= targetSpeed // 找下界，用 >=*

*// 速度与时间成反比，因此判断条件等同于：*

**if** TotalTime(piles, mid) <= H {

right = mid - 1

} **else** {

left = mid + 1

}

}

**return** left

}

**func** **TotalTime**(piles []**int**, k **int**) **int** {

time := 0

**for** \_, v := **range** piles {

time += (v+k-1)/k *// 向上取整*

}

**return** time

}

**func** **MaxOf**(nums []**int**) **int** {

m := -1 << 63

**for** \_, v := **range** nums {

m = Max(m, v)

}

**return** m

}

**func** **Max**(a, b **int**) **int** {

**if** a > b {

**return** a

}

**return** b

}

## 1011. 在 D 天内送达包裹的能力

这道题是 [LeetCode 1011 题](https://leetcode-cn.com/problems/capacity-to-ship-packages-within-d-days/)：

传送带上的包裹必须在 D 天内从一个港口运送到另一个港口。

传送带上的第 i 个包裹的重量为 weights[i]。每一天，我们都会按给出重量的顺序往传送带上装载包裹。我们装载的重量不会超过船的最大运载重量。

返回能在 D 天内将传送带上的所有包裹送达的船的最低运载能力。

本题和 875 题很像，要返回能在 D 天内将传送带上的所有包裹送达的船的**最低运载能力**，这就相当于是在找下界，可以直接套用模板代码：

1. left、right 表示运载能力，其初值为 MaxOf(weights)、SumOf(weights)。原因：货物无法拆分为更小单位，故最小运载能力是每件货物的最大重量，最大运载能力是货物重量总和，即一批运走
2. 「运载能力」和「天数」成反比，因此判断条件写为 TotalTime(weights, mid) <= D，等同于 mid >= targetCapacity，其中 targetCapacity 是「恰好」在 D 天运送完所有包裹的运载能力

完整代码如下：

*// 这里直接套用模板代码*

**func** **shipWithinDays**(weights []**int**, D **int**) **int** {

left, right := MaxOf(weights), SumOf(weights)

**for** left <= right {

mid := left + (right-left)>>1 *// mid 的含义是「运载能力」*

**if** TotalTime(weights, mid) <= D {

right = mid - 1

} **else** {

left = mid + 1

}

}

**return** left

}

**func** **TotalTime**(weights []**int**, capacity **int**) **int** {

time := 1 *// 初始化为第一天*

c := capacity

i := 0

**for** i < len(weights) {

**if** c >= weights[i] {

c -= weights[i] *// 这一天还可以装*

i++

} **else** {

time++ *// 这一天已经装满了，在下一天再装*

c = capacity

}

}

**return** time

}

**func** **MaxOf**(nums []**int**) **int** {

m := -1 << 63

**for** \_, v := **range** nums {

m = Max(m, v)

}

**return** m

}

**func** **SumOf**(nums []**int**) **int** {

m := 0

**for** \_, v := **range** nums {

m += v

}

**return** m

}

**func** **Max**(a, b **int**) **int** {

**if** a > b {

**return** a

}

**return** b

}

个人笔记

## 总结

本文通过两个实际问题，深入讲解了如何运用二分查找模板代码。我们发现，**模板代码可以进一步优化为如下形式，从而适用于任何「查找下界」的问题**：

func LowerBound(nums []int, target int) int {

left, right := MIN, MAX // 最小值、最大值

for left <= right {

mid := left + (right-left) >> 1

if IS\_OK { // 这里的判断条件与题目要求一致

right = mid - 1

} else {

left = mid + 1

}

}

return left

}

上面的代码中有**三处变量 MIN、MAX、IS\_OK，分别表示搜索区间的最小值、最大值、判断条件**，根据题目要求填写：

* [LeetCode 34 题](https://leetcode-cn.com/problems/find-first-and-last-position-of-element-in-sorted-array/)：查找满足 x >=target 的第一个位置，故 MIN、MAX、IS\_OK 分别为 0、n-1、nums[mid] >= target
  + 如果要查找满足 x >=target 的第一个位置，IS\_OK 变为 nums[mid] > target
* [LeetCode 875 题](https://leetcode-cn.com/problems/koko-eating-bananas/)：查找可以在 H 小时内吃掉所有香蕉的最小速度，故 MIN、MAX、IS\_OK 分别为 1、MaxOf(piles)、TotalTime(piles, mid) <= H
* [LeetCode 1011 题](https://leetcode-cn.com/problems/capacity-to-ship-packages-within-d-days/)：查找可以在 D 天内运损完所有货物的最小运载能力，故 MIN、MAX、IS\_OK 分别为 MaxOf(weights)、SumOf(weights)、TotalTime(weights, mid) <= D

**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。**