二分搜索 one pager

MLE 算法指北

2025 年 2 月 6 日

1 二分搜索的基本框架

二分搜索（Binary Search）是一种高效的搜索算法，适用于单调性问题（单调递增或单调递减）。其时间复杂度为 $O(\log n)$。

1.1 标准二分搜索模板

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = left + (right - left) // 2  # 防止溢出
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

- 确定二分范围
- 编写判断函数
- 根据判断结果调整边界
- 处理边界情况

2 经典应用场景

二分搜索不仅用于查找某个具体值，还可以解决查找边界、优化问题等。以下是几种常见应用场景。

2.1 查找第一个或最后一个满足条件的位置

应用场景：查找边界问题，如在有重复元素的数组中找到目标值的最左或最右索引。

示例题：

- LeetCode 34: 在排序数组中查找元素的第一个和最后一个位置
- LeetCode 278: 第一个错误的版本

模板代码（查找第一个大于等于 target 的索引）：

```
def lower_bound(arr, target):
    left, right = 0, len(arr)
    while left < right:
        mid = left + (right - left) // 2
        if arr[mid] >= target:
            right = mid
        else:
            left = mid + 1
    return left
```

2.2 求满足条件的最小值（答案二分）

应用场景：最小化最大值、最大化最小值等优化问题。

示例题：

- LeetCode 410: 分割数组的最大值
- LeetCode 1011: 在 D 天内送达包裹的能力
- LeetCode 875: 爱吃香蕉的珂珂

模板代码（求最小满足条件的值）：

```
def min_satisfying_value(condition):
    left, right = 0, 10**9  # 根据题目设置边界
    while left < right:
        mid = left + (right - left) // 2
        if check(mid):  # 检查 mid 是否满足条件
            right = mid  # 尝试更小的解
        else:
            left = mid + 1
    return left
```

2.3 二分实数（求平方根）

应用场景：当答案是一个实数时，需要二分到一定精度。

示例题：

- LeetCode 69: x 的平方根
- LeetCode 1201: 丑数 III
- LeetCode 1482: 制作 m 束花所需的最短天数

模板代码（求平方根，二分实数）：

```
def sqrt_binary_search(x, precision=1e-6):
    left, right = 0, max(1, x)
    while right - left > precision:
        mid = (left + right) / 2
        if mid * mid > x:
            right = mid
        else:
            left = mid
    return left
```

3 通用方法论

1. **确定二分的范围**：
   - 搜索具体值：通常是 $[0, n-1]$。
   - 查找边界：可能是 $[0, n]$（不包含右端点）。
2. **编写判断函数**：
   - 例如 ‘check(mid)‘ 判断 mid 是否满足条件。
3. **根据 ‘check(mid)‘ 调整边界**：
   - 若 ‘check(mid)‘ 为 ‘True‘，尝试更小的解，收缩右边界（‘right = mid‘）。
   - 若 ‘check(mid)‘ 为 ‘False‘，尝试更大的解，扩大左边界（‘left = mid + 1‘）。
4. **注意边界条件**：
   - ‘while left <= right‘ 用于搜索具体值；
   - ‘while left < right‘ 用于查找最小/最大满足条件的值；
   - ‘mid = left + (right - left) // 2‘ 防止溢出；
   - 可能需要 ‘mid = left + (right - left + 1) // 2‘，用于向上取整。