

二分搜索

二分搜索不好记，左右边界让人迷；
 小于等于变小于，mid加一再减一；
 搜索一个元素时，搜索区间两端闭；
 while条件带等号，否则需要打补丁；
 if相等就返回，其他事情甭操心；
 mid必须加减一，因为区间两端闭；
 while结束就凉凉，凄凄惨惨返-1；
 搜索左右边界时，搜索区间要阐明；
 左闭右开最常见，其余逻辑自明白；
 while要用小于号，这样才能不漏掉；
 if相等别返回，利用mid锁边界；
 mid加一或减一，要看区间开或闭；
 while结束不算完，结果记得要返回；
 索引可能出边界，if检查报平安。

二分查找框架

```
int binarySearch(std::vector<int> &nums, int target) {
    int left = 0, right = ...;
    while(...) {
        int mid = left + (right - left) / 2;
        if(nums[mid] == target) {
            ...
        } else if (nums[mid] > target) {
            right = ....;
        } else if (nums[mid] < target) {
            left = ...;
        }
    }

    return ...;
}
```

二分搜索逻辑统一：

1. 查找元素

因为我们初始化 `right = nums.length - 1` 所以决定了我们的「搜索区间」是 `[left, right]` 所以决定了 `while (left <= right)` 同时也决定了 `left = mid + 1` 和 `right = mid - 1` 因为我们只需找到一个 `target` 的索引即可 所以当 `nums[mid] == target` 时可以立即返回

2. 寻找左侧边界的二分查找

因为我们初始化 `right = nums.length` 所以决定了我们的「搜索区间」是 `[left, right)` 所以决定了 `while (left < right)` 同时也决定了 `left = mid + 1` 和 `right = mid` 因为我们需找到 `target` 的最左侧索引 所以当 `nums[mid] == target` 时不要立即返回 而要收紧右侧边界以锁定左侧边界

3. 寻找右侧边界的二分查找

因为我们初始化 `right = nums.length` 所以决定了我们的「搜索区间」是 `[left, right)` 所以决定了 `while (left < right)` 同时也决定了 `left = mid + 1` 和 `right = mid` 因为我们需找到 `target` 的最右侧索引 所以当 `nums[mid] == target` 时不要立即返回 而要收紧左侧边界以锁定右侧边界 又因为收紧左侧边界时必须 `left = mid + 1` 所以最后无论返回 `left` 还是 `right`, 必须减一

搜索区间两端封闭的框架:

1. 查找元素

```
int binarySearch(std::vector<int> &nums, int target) {
    int left = 0, right = nums.size() - 1;
    while(left <= right) {
        int mid = left + (right - left) / 2;
        if(target == nums[mid]) {
            return mid;
        } else if(target > nums[mid]) {
            left = mid + 1;
        } else if(target < nums[mid]) {
            right = mid - 1;
        }
    }

    return -1;
}
```

2. 查找左边界

```
int left_bound(std::vector<int> &nums, int target) {
    int left = 0, right = nums.size() - 1;
    while(left <= right) {
        int mid = left + (right - left) / 2;
        if(target == nums[mid]) {
            right = mid - 1;
        } else if(target < nums[mid]) {
            left = mid + 1;
        } else if(target > nums[mid]) {
            right = mid - 1;
        }
    }

    if(left >= nums.size() || nums[left] != target) {
        return -1;
    }
    return left;
}
```

3. 寻找右边界

```
int right_bound(std::vector<int> &nums, int target){
    int left = 0, right = nums.size() - 1;
    while(left <= right) {
        int mid = left + (right - left) / 2;
        if(target == nums[mid]) {
            left = mid + 1;
        } else if(target < nums[mid]) {
            left = mid + 1;
        } else if(target > nums[mid]) {
            right = mid - 1;
        }
    }

    if(right < 0 || nums[right] != target) {
        return -1;
    }
    return mid;
}
```