”吃透“二分查找

二分查找要考虑的几个问题：

1：大前提：有序数组

2：数组中无重复元素，保证搜索的唯一性

3：到底是 **`while(left < right)`** 还是 **`while(left <= right)`**

4：到底是**`right = middle`**呢，还是要**`right = middle - 1`**

二分法的第一种写法：

左闭右闭即[left, right]；

- while (left <= right) 要使用 <= ，因为left == right是有意义的，所以使用 <=
- if (nums[middle] > target) right 要赋值为 middle - 1，因为当前这个nums[middle]一定不是target，那么接下来要查找的左区间结束下标位置就是 middle - 1

```
// 版本一
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int left = 0;
        int right = nums.size() - 1; // 定义target在左闭右闭的区间里，[left, right]
        while (left <= right) { // 当left==right，区间[left, right]依然有效，所以用 <=
            int middle = left + ((right - left) / 2);// 防止溢出 等同于(left + right)/2
            if (nums[middle] > target) {
                right = middle - 1; // target 在左区间，所以[left, middle - 1]
            } else if (nums[middle] < target) {
                left = middle + 1; // target 在右区间，所以[middle + 1, right]
            } else { // nums[middle] == target
                return middle; // 数组中找到目标值，直接返回下标
            }
        }
        // 未找到目标值
        return -1;
```

```
    }
};
```

二分查找的另一种写法：

左闭右开即[left, right)

- while (left < right)，这里使用 < ,因为left == right在区间[left, right)是没有意义的
- if (nums[middle] > target) right 更新为 middle，因为当前nums[middle]不等于target，去左区间继续寻找，而寻找区间是左闭右开区间，所以right更新为middle，即：下一个查询区间不会去比较nums[middle]

```
// 版本二
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int left = 0;
        int right = nums.size(); // 定义target在左闭右开的区间里，即：[left, right)
        while (left < right) { // 因为left == right的时候，在[left, right)是无效的空间，所以使用 <
            int middle = left + ((right - left) >> 1);// 防止溢出 等同于(left + right)/2
            if (nums[middle] > target) {
                right = middle; // target 在左区间，在[left, middle)中
            } else if (nums[middle] < target) {
                left = middle + 1; // target 在右区间，在[middle + 1, right)中
            } else { // nums[middle] == target
                return middle; // 数组中找到目标值，直接返回下标
            }
        }
        // 未找到目标值
        return -1;
    }
};
```

二分法的其他应用：寻找不大于target的最大数或大于target的最小数

左闭右闭区间：

```
class Solution{
    public:
    int search_max(vector<int>&nums,int target){
        //左闭右闭区间，【left,right】
```

```
        int left=0,right=nums.size()-1;
        int pos1=-1, pos2=-1;//pos1表示的是不大于target的最大数的下标，当该数不存在时，下标为-1
        //pos2表示的是不大于target的最大数的下标，当该数不存在时，下标为-1
        while(left<=right){//此时left==right是由意义的
            int middle=(left+right)/2;
            if(nums[middle]<target){
                pos1=middle;//pos1一定是在小于target处取到
                left=middle+1;
            }else if(nums[middle]>=target){
                if(nums[middle]>target){
                    pos2=middle;//pos2一定是在大于target处取到
                }
                right=middle-1;
            }
        }
    }
}
```

左闭右开区间：

```
class Solution{
    public:
    int search_max(vector<int>&nums,int target){
        //左闭右闭区间，[left,right）
        int left=0,right=nums.size();
        int pos1=-1, pos2=-1;//pos1表示的是不大于target的最大数的下标，当该数不存在时，下标为-1
        //pos2表示的是不大于target的最大数的下标，当该数不存在时，下标为-1
        while(left<right){//此时left==right是无意义的
            int middle=(left+right)/2;
            if(nums[middle]<target){
                pos1=middle;//pos1一定是在小于target处取到
                left=middle+1;
            }else if(nums[middle]>=target){
                if(nums[middle]>target){
                    pos2=middle;//pos2一定是在大于target处取到
                }
                right=middle;
            }
        }
    }
}
```

含有重复数字求上下边界：

题目描述：

34. 在排序数组中查找元素的第一个和最后一个位置

难度 中等 1123

给定一个按照升序排列的整数数组 `nums`，和一个目标值 `target`。找出给定目标值在数组中的开始位置和结束位置。

如果数组中不存在目标值 `target`，返回 `[-1, -1]`。

进阶：

- 你可以设计并实现时间复杂度为 `O(log n)` 的算法解决此问题吗？

示例 1：

```
输入：nums = [5,7,7,8,8,10], target = 8
输出：[3,4]
```

含有重复数字的有序数组中求目标数字下标边界：

```
// 辅函数-求右边界
int lower_bound(vector<int> &nums, int target) {
int l = 0, r = nums.size(), mid;//左闭右开区间
while (l < r) {
    mid = (l + r) / 2;
    if (nums[mid] >= target) {
        r = mid;
        } else {
            l = mid + 1;
        }
    }
    return l;
}
```

```
// 辅函数--求左边界

int upper_bound(vector<int> &nums, int target) {
```

```
int l = 0, r = nums.size(), mid;
while (l < r) {
    mid = (l + r) / 2;
    if (nums[mid] > target) {
        r = mid;
        } else {
            l = mid + 1;
        }
    }
    return l;
}
```

多段有序数组应用二分查找：

题目描述：

81. Search in Rotated Sorted Array II (Medium)

题目描述

一个原本增序的数组被首尾相连后按某个位置断开（如 [1,2,2,3,4,5] → [2,3,4,5,1,2]，在第一位和第二位断开），我们称其为旋转数组。给定一个值，判断这个值是否存在于这个为旋转数组中。

输入输出样例

输入是一个数组和一个值，输出是一个布尔值，表示数组中是否存在该值。

```
Input: nums = [2,5,6,0,0,1,2], target = 0
Output: true
```

题解

即使数组被旋转过，我们仍然可以利用这个数组的递增性，使用二分查找。对于当前的中点，如果它指向的值小于等于右端，那么说明右区间是排好序的；反之，那么说明左区间是排好序的。如果目标值位于排好序的区间内，我们可以对这个区间继续二分查找；反之，我们对于另一半区间继续二分查找。

注意，因为数组存在重复数字，如果中点和左端的数字相同，我们并不能确定是左区间全部相同，还是右区间完全相同。在这种情况下，我们可以简单地将左端点右移一位，然后继续进行二分查找。

```
class Solution {
public:
    bool search(vector<int>& nums, int target) {
        //变形题
        int left=0,right=nums.size()-1;
        while(left<=right){
```

```
            int mid=(left+right)/2;
            if(nums[mid]==target)return true;
            if(nums[mid]==nums[right]){
                //无法判断那个区间时正序
                --right;//消除相等的数带来的干扰
            }else if(nums[mid]<=nums[right]){
                //右区间是正序
                //修改一下区间即可
                if(nums[mid]<target&&target<=nums[right]){
                    left=mid+1;
                }else right=mid-1;
            }else if(nums[mid]>=nums[left]){
                //左区间是正序
                //修改一下区间
                if(nums[mid]>target&&target>=nums[left]){
                    right=mid-1;
                }else left=mid+1;
            }

        }
        return false;
    }
};
```