二分查找法的第一种——闭区间

In [15]:

```python
### 查找不重复元素中的target

def non_search1(nums,target):
    left,right = 0,len(nums)-1
    while left <= right:
        mid = left + (right-left)//2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1   # 永远不变
        elif nums[mid] > target:
            right = mid -1
    return -1



# 找右缩左
def left_search1(nums,target):
    left,right = 0,len(nums)-1
    while left <= right:
        mid = left + (right-left)//2
        if nums[mid] == target:
            right = mid -1   # 同 nums[mid] > target这种情况，不断压缩右边
        elif nums[mid] < target:
            left = mid + 1   # 永远不变
        elif nums[mid] > target:
            right = mid -1   # 压缩右边界

    # 越界条件
    if left == len(nums) or nums[left] != target:
        return -1
    return left

# 找左缩右
def right_search1(nums,target):
    left,right = 0,len(nums)-1
    while left <= right:
        mid = left + (right-left)//2
        if nums[mid] == target:
            left = mid + 1   # 同 nums[mid] < target这种情况,不断压缩左边
        elif nums[mid] < target:
            left = mid + 1   # 永远不变
        elif nums[mid] > target:
            right = mid -1

    # 越界条件
    if right < 0 or nums[right] != target:
        return -1
    return right
```

二分查找法的第二种——开区间

In [8]:

```python
### 查找不重复元素中的target

def non_search2(nums,target):
    left,right = 0,len(nums)
    while left < right:      # right 变就变
        mid = left + (right-left)//2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1   # 永远不变
        elif nums[mid] > target:
            right = mid      # right 变就变
    return -1

def left_search2(nums,target):
    left,right = 0,len(nums)
    while left < right:
        mid = left + (right-left)//2
        if nums[mid] == target:
            right = mid    # 同 nums[mid] > target这种情况，不断压缩右边
        elif nums[mid] < target:
            left = mid + 1   # 永远不变
        elif nums[mid] > target:
            right = mid
    return left  # 返回左边界

def right_search2(nums,target):
    left,right = 0,len(nums)
    while left < right:
        mid = left + (right-left)//2
        if nums[mid] == target:
            left = mid + 1   # 同 nums[mid] < target这种情况,不断压缩左边
        elif nums[mid] < target:
            left = mid + 1   # 永远不变
        elif nums[mid] > target:
            right = mid
    return right-1  # 记得-1 ，因为是开区间
```

In [ ]:

```python
# 875 二分吃香蕉
def minEatingSpeed(self, piles: List[int], h: int) -> int:
    # 二分法,速度有序
    def canFindish(piles,speed):
        # H 小时内能吃完
        time = 0
        for n in piles:
            time += (n//speed)+(1 if n%speed > 0 else 0 )  # 向上取整
        return time <= h
    left = 1 # 速度从1开始
    right = max(piles)
    while left <= right:
        mid = left + (right-left)//2
        if canFindish(piles,mid):
            right = mid -1
        else:
            left = mid + 1
    return left

# 1011 货运船
def shipWithinDays(self, weights: List[int], days: int) -> int:
        def canFindish(weights,D,cap):
        # H 小时内能吃完
            days = 1
            current = 0
            for weight in weights:
                current += weight
                if current > cap:
                    days += 1
                    current = weight
            return days <= D
        left = max(weights) # 速度从1开始
        right = sum(weights)
        while left <= right:
            mid = left + (right-left)//2
            if canFindish(weights,days,mid):
                right = mid -1
            else:
                left = mid + 1
        return left
```