二分专题

二分查找总结

二分写法注意点

1. 循环退出条件，注意是 low <= high，而不是 low < high。
2. mid 的取值，mid := low + (high-low)>>1
3. low 和 high 的更新。low = mid + 1，high = mid - 1

普通的二分查找

非递归实现

```
public int binarySearch(int[] a, int n, int value) {
  int low = 0;
  int high = n - 1;

  while (low <= high) {
    int mid = (low + high) / 2;
    if (a[mid] > value) {
       high = mid - 1;
     } else if (a[mid] < value) {
      low = mid + 1;
    } else {
      return mid;
    }
  }

  return -1;
}
```

递归实现

```
// 二分查找的递归实现
public int bsearch(int[] a, int n, int val) {
  return bsearchInternally(a, 0, n - 1, val);
}

private int bsearchInternally(int[] a, int low, int high, int value) {
  if (low > high) return -1;
  int mid =  low + ((high - low) >> 1);
  if (a[mid] > value) {
    return bsearchInternally(a, low, mid-1, value);
  } else if (a[mid] < value) {
    return bsearchInternally(a, mid+1, high, value);
```

```
  } else {
    return mid;
  }
}
```

二分查找的简单变形问题

- 查找第一个值等于给定值的元素
- 查找最后一个值等于给定值的元素
- 查找第一个大于等于给定值的元素
- 查找最后一个小雨等于给定值的元素

查找第一个值等于给定值的元素

```
public int binarySearch(int[] a, int n, int value) {
  int low = 0;
  int high = n - 1;
  while (low <= high) {
    int mid =  low + ((high - low) >> 1);
    if (a[mid] > value) {
      high = mid - 1;
    } else if (a[mid] < value) {
      low = mid + 1;
    } else {
      if ((mid == 0) || (a[mid - 1] != value)) return mid;
      else high = mid - 1;
    }
  }
  return -1;
}
```

如果 mid 等于 0，那这个元素已经是数组的第一个元素，那它肯定是我们要找的；如果 mid 不等于 0，但 a[mid]的前一个元素 a[mid-1]不等于 value，那也说明 a[mid]就是我们要找的第一个值等于给定值的元素。

查找最后一个值等于给定值的元素

```
public int binarySearch(int[] a, int n, int value) {
  int low = 0;
  int high = n - 1;
  while (low <= high) {
    int mid =  low + ((high - low) >> 1);
    if (a[mid] > value) {
      high = mid - 1;
    } else if (a[mid] < value) {
      low = mid + 1;
    } else {
      if ((mid == n - 1) || (a[mid + 1] != value)) return mid;
      else low = mid + 1;
```

```
    }
  }
  return -1;
}
```

如果 a[mid]这个元素已经是数组中的最后一个元素了，那它肯定是我们要找的；如果 a[mid]的后一个元素 a[mid+1]不等于 value，那也说明 a[mid]就是我们要找的最后一个值等于给定值的元素。

查找第一个大于等于给定值的元素

```
public int binarySearch(int[] a, int n, int value) {
  int low = 0;
  int high = n - 1;
  while (low <= high) {
    int mid =  low + ((high - low) >> 1);
    if (a[mid] >= value) {
      if ((mid == 0) || (a[mid - 1] < value)) return mid;
      else high = mid - 1;
    } else {
      low = mid + 1;
    }
  }
  return -1;
}
```

查找最后一个小雨等于给定值的元素

```
public int binarySearch(int[] a, int n, int value) {
  int low = 0;
  int high = n - 1;
  while (low <= high) {
    int mid =  low + ((high - low) >> 1);
    if (a[mid] > value) {
      high = mid - 1;
    } else {
      if ((mid == n - 1) || (a[mid + 1] > value)) return mid;
      else low = mid + 1;
    }
  }
  return -1;
}
```

刷题指南

704. 二分查找（简单）	
34. 在排序数组中查找元素的第一个和最后一个位置（中等）	
搜索旋转排序数组（中等）	
搜索旋转排序数组 II（中等）	
153. 寻找旋转排序数组中的最小值（中等）	
寻找旋转排序数组中的最小值 II（中等）	
852. 山脉数组的峰顶索引（简单）	
1095. 山脉数组中查找目标值（中等）	
4. 寻找两个有序数组的中位数（困难）	
658. 找到 K 个最接近的元素（中等）	

搜索选择旋转的排序数组

```
class Solution {
    public int search(int[] nums, int target) {
        int low = 0;
        int high = nums.length - 1;
        while(low <= high){
            int mid = low + ((high - low)>>1);
            if(nums[mid] == target) return mid;
            if(nums[low] <= nums[mid]){
                if(nums[low] <= target && target < nums[mid]){
                    high = mid - 1;
                }else{
                    low = mid + 1;
                }
            }else{
                if(nums[mid] < target && target <= nums[high]){
                    low = mid + 1;
                }else{
                    high = mid - 1;
                }

            }
    }
    return -1;
    }
}
```

搜索选择旋转的排序数组II

```
class Solution {
    public boolean search(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;
        while(left <= right){
            int mid = left + (right-left)/2;
            if(nums[mid]==target) return true;
            // 注意left = mid 的情况
            if(nums[left] == nums[mid]){
                left ++;
                continue;
            }
            if(nums[left] < nums[mid]){
                if(nums[left] <= target && target < nums[mid]){
                    right = mid - 1;
                }else{
                    left = mid + 1;
                }
            }else{
                if(nums[mid] < target && target <= nums[right]){
                    left = mid + 1;
                }else{
                    right = mid - 1;
                }
            }
        }
        return false;
    }
}
```

搜索选择数组中的最小值

```
class Solution {
    public int findMin(int[] nums) {
        int left = 0;
        int right = nums.length - 1;
        int mid = 0;
        while(left <= right){
            mid = left + (right - left)/2;
            if(nums[mid]<nums[right]){
                right = mid;
            }else{
                left = mid + 1;
            }
        }
        return nums[mid];
```

```
    }
}
```

搜索选择数组中的最小值 II

```
class Solution {
    public int findMin(int[] nums) {
        int left = 0;
        int right = nums.length - 1;
        while(left < right){
            int mid = left + (right - left)/2;
            // left > right
            // 1. right < left < mid    最小值右边
            // 2. mid < right < left    最小值在左边
            // 3. left < mid < right    最小值在左边
            if(nums[mid] < nums[right]){
                // 左边
                right = mid;
            }else if(nums[mid] > nums[right]){
                left = mid + 1;
            }else{
                right --;
            }
        }
        return nums[left];
    }
}
```

山脉数组的峰顶索引

```
class Solution {
    public int peakIndexInMountainArray(int[] arr) {
        int left = 0;
        int right = arr.length - 1;
        while(left < right){
            int mid = left + (right - left)/2;
            // 1234 5 6321
            // left mid right
            if(arr[mid]>arr[mid+1] && arr[mid]>arr[mid-1]) return mid;
            if(arr[mid] > arr[mid-1] ){
                // 右边
                left = mid + 1;
            }else{
                right = mid;
            }
        }
```

```
        return -1;
    }
}
```

山脉元素中查找目标值

```
/**
 * // This is MountainArray's API interface.
 * // You should not implement it, or speculate about its implementation
 * interface MountainArray {
 *     public int get(int index) {}
 *     public int length() {}
 * }
 */

class Solution {
    public int findInMountainArray(int target, MountainArray mountainArr) {
        int peek = this.findPeek(mountainArr);
        int r1 = this.asSort(peek, target, mountainArr);
        if(r1 != -1){
            return r1;
        }
        int r2 = this.descSort(peek, target, mountainArr);
        if(r2 != -1){
            return r2;
        }
        return -1;
    }
    public int findPeek(MountainArray mountainArr){
        int left = 0;
        int right = mountainArr.length() - 1;
        while(left <= right){
            int mid = left + (right - left)/2;
            // 1234321
            int tempMid = mountainArr.get(mid);
            int tempMid2 = mountainArr.get(mid-1);
            if(tempMid>tempMid2
                && tempMid>mountainArr.get(mid+1))
                return mid;
            if(tempMid>tempMid2){
                left = mid + 1;
            }else{
                right = mid;
            }
        }
        return -1;
    }
```

```
    public int asSort(int n, int target, MountainArray mountainArr){
        int left = 0;
        int right = n;
        while(left <= right){
            int mid = left + (right - left)/2;
            // 12345
            if(mountainArr.get(mid) == target){
                return mid;
            }
            else if(mountainArr.get(mid) > target){
                right = mid - 1;
            }else{
                left = mid + 1;
            }
        }
        return -1;
    }

        public int descSort(int n, int target, MountainArray mountainArr){
        int left = n;
        int right = mountainArr.length() - 1;
        while(left <= right){
            int mid = left + (right - left)/2;
            // 54321
            if(mountainArr.get(mid) == target){
                return mid;
            }
            else if(mountainArr.get(mid) > target){
                left = mid + 1;
            }else{
                right = mid - 1;
            }
        }
        return -1;
    }
}
```

在一个有范围的区间里搜索一个整数

69. 平方根（简单）
1300. 转变数组后最接近目标值的数组和

平方根

```
class Solution {
    public int mySqrt(int x) {
        // 效率太低，改用二分法
        // int i = 1;
        // while(true){
        //     if(x/i < i){
        //         break;
        //     }
        //     i++;
        // }
        // return i-1;

        if(x<=1) return x;
        int l = 1, r = x/2, ans = -1;
        while (l <= r) {
            int mid = l + (r - l) / 2;
            if (mid <= x/mid) {
                ans = mid;
                l = mid + 1;
            }
            else {
                r = mid - 1;
            }
        }
        return ans;
    }
}
```

转变数组后最接近目标值的数组合

```
public class Solution {

    public int findBestValue(int[] arr, int target) {
        int left = 0;
        int right = 0;
        // 遍历寻找最大值
        for (int num : arr) {
            right = Math.max(right, num);
        }
        while (left < right) {
            int mid = left + (right - left) / 2;
            int sum = calculateSum(arr, mid);
            // 以target > sum 去做，后面再用left - 1,left 比较以确定最小值
```

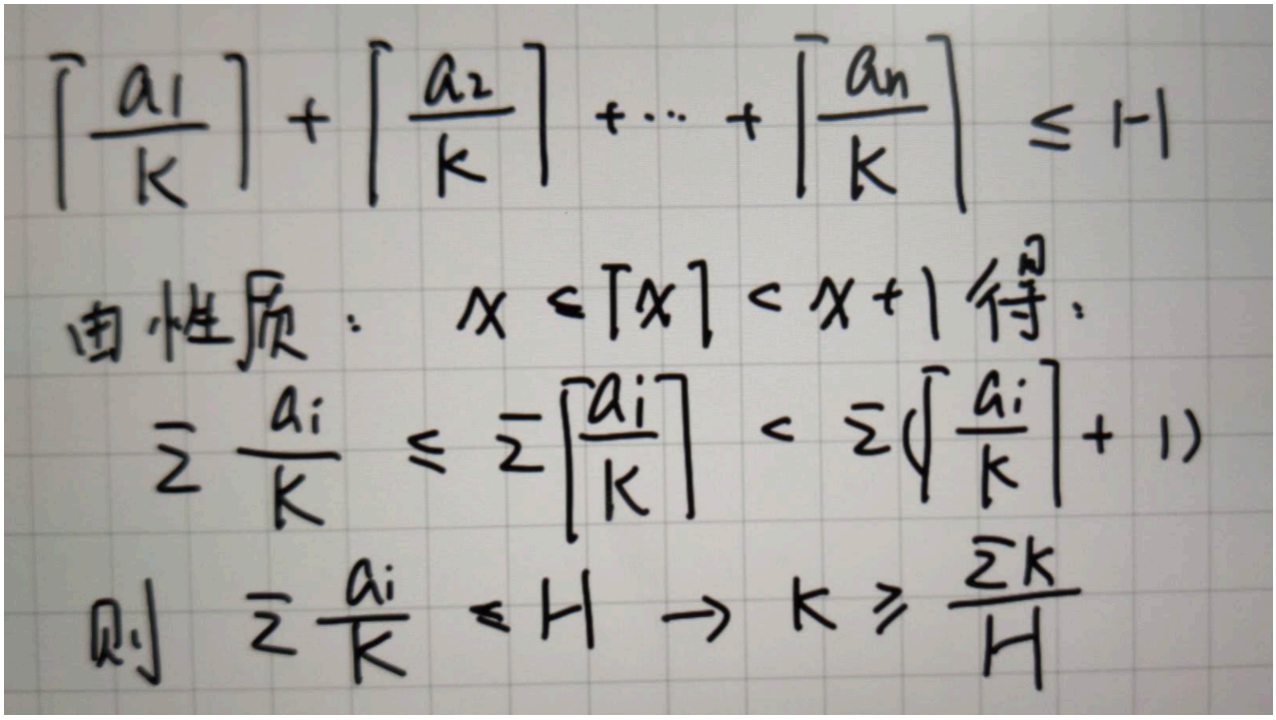
```
            if (sum < target) {
                // 严格小于的一定不是解
                left = mid + 1;
            } else {
                right = mid;
            }
        }

        // 比较阈值线分别定在 left - 1 和 left 的时候与 target 的接近程度
        int sum1 = calculateSum(arr, left - 1);
        int sum2 = calculateSum(arr, left);
        if (target - sum1 <= sum2 - target) {
            return left - 1;
        }
        return left;
    }

    private int calculateSum(int[] arr, int threshold) {
        // 计算和
        int sum = 0;
        for (int num : arr) {
            sum += Math.min(num, threshold);
        }
        return sum;
    }
}
```

复杂的二分查找问题

875. 爱吃香蕉的珂珂（中等）
1482. 制作 m 束花所需的最少天数（中等）
1552. 两球之间的磁力（中等）



爱吃香蕉的珂珂

```
public class Solution {

    public int minEatingSpeed(int[] piles, int H) {
        int maxVal = 1;
        for (int pile : piles) {
            maxVal = Math.max(maxVal, pile);
        }

        // 速度最小的时候，耗时最长
        int left = 1;
        // 速度最大的时候，耗时最短
        int right = maxVal;

        while (left < right) {
            int mid = left + (right - left) / 2;

            if (calculateSum(piles, mid) > H) {
                // 耗时太多，说明速度太慢了，下一轮搜索区间在
                // [mid + 1, right]
                left = mid + 1;
            } else {
                right = mid;
            }
        }
        return left;
```

```
    }

    private int calculateSum(int[] piles, int speed) {
        int sum = 0;
        for (int pile : piles) {
            // 上取整可以这样写
            sum += (pile + speed - 1) / speed;
        }
        return sum;
    }
}
```