二分算法刷题路线

力扣

二分查找

给定一个 `n` 个元素有序的（升序）整型数组 `nums` 和一个目标值 `target` ，写一个函数搜索 `nums` 中的 `target`，如果 `target` 存在返回下标，否则返回 `-1`。

你必须编写一个具有 $O(\log n)$ 时间复杂度的算法。

示例 1:

> 输入: `nums = [-1,0,3,5,9,12]`, `target = 9`
> 输出: 4
> 解释: 9 出现在 `nums` 中并且下标为 4

示例 2:

> 输入: `nums = [-1,0,3,5,9,12]`, `target = 2`
> 输出: -1
> 解释: 2 不存在 `nums` 中因此返回 -1

提示:

1. 你可以假设 `nums` 中的所有元素是不重复的。
2. `n` 将在 $[1, 10000]$ 之间。
3. `nums` 的每个元素都将在 $[-9999, 9999]$ 之间。

```cpp
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int left = 0;                       // 搜索区间的左边界
        int right = nums.size()-1;          // 搜索区间的右边界

        while(left<=right){                 // 当区间有效时继续搜索
            int mid = left + (right-left)/2; // 计算中间位置，防止溢出
            if(nums[mid] == target){        // 找到目标值
                return mid;                 // 返回索引
            }else if(nums[mid] < target){ // 目标值在右侧
                left = mid+1;               // 收缩左边界
            }else{                          // 目标值在左侧
                right = mid-1;              // 收缩右边界
            }
        }
        return -1;                          // 未找到目标值
    }
};
```

> 计算中间位置

$$left + \frac{right - left}{2} = \frac{2 \times left + right - left}{2} = \frac{left + right}{2}$$

猜数字大小

我们正在玩猜数字游戏。猜数字游戏的规则如下：

我会从 `1` 到 `n` 随机选择一个数字。请你猜出这个数字是多少。（我选的数字在整个游戏过程中保持不变）。

如果你猜错了，我会告诉你，我选出的数字比你猜测的数字大了还是小了。

你可以通过调用一个预先定义好的接口 `int guess(int num)` 来获取猜测结果，返回值一共有三种可能的情况：

- `-1`：你猜的数字比我选出的数字大（即 `num > pick`）。
- `1`：你猜的数字比我选出的数字小（即 `num < pick`）。
- `0`：你猜的数字与我选出的数字相等（即 `num == pick`）。

返回我选出的数字。

示例 1：

> 输入：`n = 10, pick = 6`
> 输出：`6`

示例 2：

> 输入：`n = 1, pick = 1`
> 输出：`1`

示例 3：

> 输入：`n = 2, pick = 1`
> 输出：`1`

提示：

- $1 \leq n \leq 2^{31} - 1$
- $1 \leq pick \leq n$

```cpp
/**
 * Forward declaration of guess API.
 * @param  num   your guess
 * @return       -1 if num is higher than the picked number
 *                1 if num is lower than the picked number
 *               otherwise return 0
 * int guess(int num);
 */

class Solution {
public:
    int guessNumber(int n) {
        int left = 1;
        int right = n;

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

```
            if(res == 0){
                return mid;
            }else if(res == -1){
                right = mid-1;
            }else{
                left = mid+1;
            }
        }
        return -1;
    }
};
```

搜索插入位置

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

请必须使用时间复杂度为 $O(\log n)$ 的算法。

示例 1:

> 输入: `nums = [1,3,5,6], target = 5`
> 输出: 2

示例 2:

> 输入: `nums = [1,3,5,6], target = 2`
> 输出: 1

示例 3:

> 输入: `nums = [1,3,5,6], target = 7`
> 输出: 4

提示:

- $1 \leq nums.length \leq 10^4$
- $-10^4 \leq nums[i] \leq 10^4$
- `nums` 为 无重复元素 的 升序 排列数组
- $-10^4 \leq target \leq 10^4$

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        int left = 1;
        int right = nums.size()-1;

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

            if(nums[mid] == target){
                return mid;
            }else if(nums[mid] < target){
                left = mid+1;
            }else{
                right = mid-1;
            }
        }
```

```
        return left;
    }
};
```

return left;：逼近过程

在排序数组中查找元素的第一个和最后一个位置

给你一个按照非递减顺序排列的整数数组 `nums`，和一个目标值 `target`。请你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]`。

你必须设计并实现时间复杂度为 $O(\log n)$ 的算法解决此问题。

示例 1:

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

示例 2:

输入: `nums = [5,7,7,8,8,10], target = 6` 输出: [-1,-1]

示例 3:

输入: `nums = [], target = 0` 输出: [-1,-1]

提示:

- $0 \leq nums.length \leq 10^5$
- $-10^9 \leq nums[i] \leq 10^9$
- `nums` 是一个非递减数组
- $-10^9 \leq target \leq 10^9$

```
class Solution {
public:
    int find(vector<int>& nums, int target,bool isFist){
        int left = 0;
        int right = nums.size()-1;
        int border = -1;

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

            if(nums[mid] == target){
                border = mid;
                if(isFist){
                    right = mid-1;
                }else{
                    left = mid+1;
                }
            }else if(nums[mid]<target){
                left = mid+1;
            }else{
                right = mid-1;
            }
        }
        return border;
    }
```

```
    vector<int> searchRange(vector<int>& nums, int target) {
        int left_find = find(nums,target,true);
        int right_find = find(nums,target,false);

        return {left_find,right_find};
    }
};
```

> 两次二分

x 的平方根

给你一个非负整数 `x` ，计算并返回 `x` 的 算术平方根 。

由于返回类型是整数，结果只保留 整数部分 ，小数部分将被 舍去 。

注意： 不允许使用任何内置指数函数和算符，例如 `pow(x, 0.5)` 或者 `x ** 0.5` 。

示例 1：

> 输入： `x = 4`
> 输出： `2`

示例 2：

> 输入： `x = 8`
> 输出： `2`
> 解释： `8` 的算术平方根是 `2.82842...` ，由于返回类型是整数，小数部分将被舍去。

提示：

- $0 \leq x \leq 2^{31} - 1$

```
class Solution {
public:
    int mySqrt(int x) {
        if(x == 0){return 0;}
        if(x == 1){return 1;}

        int left = 2;
        int right = x;
        int ans = 1;

        while(left<=right){
            int mid = left+(right-left)/2;
            if((long long)mid*mid<=x){
                ans = mid;
                left = mid+1;
            }else{
                right = mid-1;
            }
        }
        return ans;
    }
};
```

第一个错误的版本

你是产品经理，目前正在带领一个团队开发新的产品。不幸的是，你的产品的最新版本没有通过质量检测。由于每个版本都是基于之前的版本开发的，所以错误的版本之后的所有版本都是错的。

假设你有 `n` 个版本 `[1, 2, ..., n]`，你想找出导致之后所有版本出错的第一个错误的版本。

你可以通过调用 `bool isBadVersion(version)` 接口来判断版本号 `version` 是否在单元测试中出错。实现一个函数来找出第一个错误的版本。你应该尽量减少对 API 的调用次数。

示例 1：

> 输入：`n = 5, bad = 4`
> 输出：`4`
> 解释：
> 调用 `isBadVersion(3) -> false`
> 调用 `isBadVersion(5) -> true`
> 调用 `isBadVersion(4) -> true`
> 所以，4 是第一个错误的版本。

示例 2：

> 输入：`n = 1, bad = 1`
> 输出：`1`

提示：

- $1 \leq bad \leq n \leq 2^{31} - 1$

```cpp
// The API isBadVersion is defined for you.
// bool isBadVersion(int version);

class Solution {
public:
    int firstBadVersion(int n) {
        int left =1;
        int right = n;
        int ans = n;

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

            if(isBadVersion(mid)){
                ans = mid;
                right = mid-1;
            }else{
                left = mid+1;
            }
        }
        return ans;
    }
};
```

搜索旋转排序数组

整数数组 `nums` 按升序排列，数组中的值 **互不相同** 。

在传递给函数之前，`nums` 在预先未知的某个下标 `k`（`0 <= k < nums.length`）上进行了 **向左旋转**，使数组变为 `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]`（下标 **从 0 开始** 计数）。例如，`[0,1,2,4,5,6,7]` 下标 `3` 上向左旋转后可能变为 `[4,5,6,7,0,1,2]` 。

给你 **旋转后** 的数组 `nums` 和一个整数 `target` ，如果 `nums` 中存在这个目标值 `target` ，则返回它的下标，否则返回 `-1` 。

你必须设计一个时间复杂度为 $O(\log n)$ 的算法解决此问题。

示例 1：

> 输入：`nums = [4,5,6,7,0,1,2], target = 0` 输出：4

示例 2：

> 输入：`nums = [4,5,6,7,0,1,2], target = 3` 输出：-1

示例 3：

> 输入：`nums = [1], target = 0` 输出：-1

提示：

- $1 \leq nums.length \leq 5000$
- $-10^4 \leq nums[i] \leq 10^4$
- `nums` 中的每个值都 独一无二
- 题目数据保证 `nums` 在预先未知的某个下标上进行了旋转
- $-10^4 \leq target \leq 10^4$

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

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

            if(nums[mid] == target){
                return mid;
            }

            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 -1;
    }
};
```

> 将数组一分为二，其中一半必定是有序的

寻找旋转排序数组中的最小值

已知一个长度为 `n` 的数组，预先按照升序排列，经由 `1` 到 `n` 次 旋转 后，得到输入数组。例如，原数组 `nums = [0,1,2,4,5,6,7]` 在变化后可能得到：

- 若旋转 `4` 次，则可以得到 `[4,5,6,7,0,1,2]`
- 若旋转 `7` 次，则可以得到 `[0,1,2,4,5,6,7]`

注意，数组 `[a[0], a[1], a[2], ..., a[n-1]]` 旋转一次 的结果为数组 `[a[n-1], a[0], a[1], a[2], ..., a[n-2]]` 。

给你一个元素值 互不相同 的数组 `nums` ，它原来是一个升序排列的数组，并按上述情形进行了多次旋转。请你找出并返回数组中的 最小元素 。

你必须设计一个时间复杂度为 $O(\log n)$ 的算法解决此问题。

示例 1：

> 输入：`nums = [3,4,5,1,2]` 输出：1
> 解释：原数组为 `[1,2,3,4,5]` ，旋转 3 次得到输入数组。

示例 2：

> 输入：`nums = [4,5,6,7,0,1,2]` 输出：0
> 解释：原数组为 `[0,1,2,4,5,6,7]` ，旋转 4 次得到输入数组。

示例 3：

> 输入：`nums = [11,13,15,17]` 输出：11
> 解释：原数组为 `[11,13,15,17]` ，旋转 4 次得到输入数组。

提示：

- `n == nums.length`
- $1 \le n \le 5000$
- $-5000 \le nums[i] \le 5000$
- `nums` 中的所有整数 互不相同
- `nums` 原来是一个升序排列的数组，并进行了 `1` 至 `n` 次旋转

```cpp
class Solution {
public:
    int findMin(vector<int>& nums) {
        int left = 0;
        int right = nums.size()-1;

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

            if(nums[mid] < nums[right]){
                right = mid;
            }else{
                left = mid+1;
            }
        }
        return nums[left];
    }
};
```

寻找峰值

峰值元素是指其值严格大于左右相邻值的元素。

给你一个整数数组 `nums`，找到峰值元素并返回其索引。数组可能包含多个峰值，在这种情况下，返回 任何一个峰值 所在位置即可。

你可以假设 `nums[-1] = nums[n] = -∞` 。

你必须实现时间复杂度为 $O(\log n)$ 的算法来解决此问题。

示例 1：

> 输入： `nums = [1,2,3,1]`
> 输出： 2
> 解释： 3 是峰值元素，你的函数应该返回其索引 2。

示例 2：

> 输入： `nums = [1,2,1,3,5,6,4]`
> 输出： 1 或 5
> 解释： 你的函数可以返回索引 1，其峰值元素为 2；或者返回索引 5，其峰值元素为 6。

提示：

- $1 \leq nums.length \leq 1000$
- $-2^{31} \leq nums[i] \leq 2^{31} - 1$
- 对于所有有效的 `i` 都有 `nums[i] != nums[i + 1]`

```cpp
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int left = 0;
        int right = nums.size()-1;

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

            if(nums[mid]<nums[mid+1]){
                left = mid+1;
            }else{
                right = mid;
            }
        }
        return left;
    }
};
```