二分法

2022年8月17日 16:04

二分法最重要的是**对区间的定义**，区间的定义是不变量。
在二分查找的过程中，保持不变量，就是在while寻找中每一次边界的处理都要坚持根据区间的定义来操作，这就是**循环不变量**规则
不变量的意义在于。在开始循环之前若target所在位置满足[left,right]或者[left,right)，那么在二分查找进行迭代的过程中，也依旧满足在该**范围内（[left,right]或者[left,right)）**查找。那么在最后一次迭代中，仍旧满足条件！
返回值只需要根据循环结束的临界情况进行调整即可！
那么根据不变量的定义（left<=right）或（left<right），我们可以判断出是[left,right]或[left,right)是如何变成空集（left穿过right？right穿过left？或者left右移与right重叠？right左移与left重叠?）

Leetcode35

35. 搜索插入位置

难度 简单 1672

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。

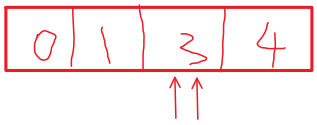
请必须使用时间复杂度为 `O(log n)` 的算法。

示例 1:

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

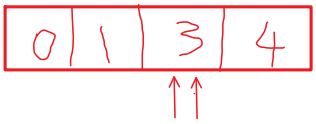
```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        int left=0,right=nums.size()-1;
        int mid;
        while(left<=right){
            mid=left+(right-left)/2;
            if(nums[mid]==target)
                return mid;
            if(nums[mid]>target)
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

考虑临界条件时（left==right）：
1.nums[mid]==target



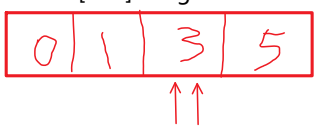
直接return mid

2.nums[mid]>target



Right=left(mid)-1=2-1=1

3.nums[mid]<target



left=right(mid)+1=2+1=3

综上所述，可见最终返回left即可

左闭右开法同理考虑临界情况（实际上此题循环不变量采用区间**[left,rigth)更为合适**）
因为可能出现nums[mid]<target<nums[mid+1]的情况，此时区间[left,right]在**最后一次迭代中会出现循环不变量为假**的情况，也就是说target不在[left,right]区间中了（此时left与right相等）。
无论是left右移与right重叠或是right左移与left重叠，均应返回left或right均可

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

Leetcode69

69. x 的平方根

难度 简单 1117

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
```

由上所述，此题的循环不变量应采取左闭右开

```cpp
class Solution {
public:
    int mySqrt(int x) {
        int left=1;
        int right=x/2+2;
        while(left<right){
            int mid=left+(right-left)/2;
            if(mid>x/mid){
                right=mid;
            }
            else if(mid<x/mid){
                left=mid+1;
            }
            else{
                return mid;
            }
        }
        return right-1;
    }
};
```

Leetcode34

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 1859

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。

此题循环不变量采取左闭右闭更为合适

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

    vector<int> searchRange(vector<int>& nums, int target) {
        int leftIdx = binarySearch(nums, target, true);
        int rightIdx = binarySearch(nums, target, false);
        if (leftIdx <= rightIdx && rightIdx < nums.size() && nums[leftIdx] == target && nums[rightIdx] == target) {
            return vector<int>{leftIdx, rightIdx};
        }
        return vector<int>{-1, -1};
    }
};
```

当nums[mid]>=target时，需要不断右移边界，来获取左边界leftIdx

注意，在最后一次迭代（循环不变量仍为真）结束后，right穿过left循环结束，因此ans=right+1

当nums[mid]>target（nums[mid]<=target）时，需要不断左移边界，来获取右边界rightIdx

注意，在最后一次迭代（循环不变量仍为真）结束后，left穿过right循环结束，因此ans=left-1；

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

    vector<int> searchRange(vector<int>& nums, int target) {
        int leftIdx = binarySearch(nums, target, true);
        int rightIdx = binarySearch(nums, target, false);
        if (leftIdx <= rightIdx && rightIdx < nums.size() && nums[leftIdx] == target && nums[rightIdx] == target) {
            return vector<int>{leftIdx, rightIdx};
        }
        return vector<int>{-1, -1};
    }
};
```

当nums[mid]>=target时，需要不断右移边界，来获取左边界leftIdx

注意，在最后一次迭代（循环不变量仍为真）结束后，right左移与left重叠，循环结束，因此ans=right

当nums[mid]>target（nums[mid]<=target）时，需要不断左移边界，来获取右边界rightIdx

注意，在最后一次迭代（循环不变量仍为真）结束后，left=mid+1与right重叠或穿过right，循环结束，因此ans=right-1；