数组

- 下标从0开始，在内存空间上是连续的

704 (https://leetcode.cn/problems/binary-search/) 二分查找 ¶

默认数组是有序数组（升序），且数组中无重复元素。（这两个条件是二分查找的前置条件，有他们就要想到二分查找）

只要提供的数组是有序的,就要想到二分查找

第一种写法:

- 定义target在[left,right]区间
- whiel(left<=right),因为left==fight是有意义的，所以使用<=
- if(nums[middle]>target) right要赋值为middle-1

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int left = 0;
        int right = nums.size()-1;
        while(left<=right){
            int middle = left + (right-left)/2; // middle一定在while中定义，因为每次middle都不一样
            if(nums[middle]>target){
                right = middle-1; // right可以取到，middle已经判断为不是目标值，所以是middle-1，如果是middle会死循环
            } else if(nums[middle]<target){
                left = middle+1;
            } else {
                return middle;
            }
        }
        return -1;
    }
};
```

- 时间复杂度：O（logn）
- 空间复杂度：O(1)

第二种写法

- 定义target在[left,right)
- while(left<right),因为left==right是没有意义的
- if(nums[middle]>target),right更新为middle，下一个区间[left,millde)不包含middle

```
class Solution {
public:
    int search(vector<int>& nums, int target) {
      int left = 0;
      int right = nums.size(); // 这里不同
      while(left<right){ // 这里不同
          int middle = left +(right-left)/2; // 等同于 int middle = left + ((right-lef
t))>>1;
          if(nums[middle]>target){
              right = middle; // 这里不同
          } else if(nums[middle]<target){
              left = middle+1;
          } else {
              return middle;
          }
```

相关题目

[35 (https://leetcode.cn/problems/search-insert-position/)](https://leetcode.cn/problems/search-insert-position/) **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
    *nums中无重复元素

    *思路：
    *在数组中插入目标值一共有四种情况：
    * 1. 目标值在数组所有元素之后
    * 2. 目标值在数组所有元素之前
    * 3. 目标值插入数组中的位置
    * 4. 目标值等于数组中的某一个元素
*/

// 暴力解法
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        for(decltype(nums.size()) i = 0;i!=nums.size();++i){
            if(nums[i]>= target){ // 包含2.3.4三种情况
                return i;
            }
        }
        return nums.size();
    }
};

// 二分法1（[left,right]）
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        int left  = 0;
        int right = nums.size()-1; //dd
        while(left<=right){ // dd
            int middle = left + (right-left)/2;
            if(nums[middle]>target){
                right = middle-1; // dd
            } else if(nums[middle]<target){
                left = middle+1;
            } else {
                return middle;
            }
        }
        //分别考虑四种情况
        //如果目标值等于数组中的某一个元素，返回middle
```

34 (https://leetcode.cn/problems/find-first-and-last-position-of-element-in-sorted-array/).在排序数组中查找元素的第一个和最后一个位置，没有目标值返回[-1,-1]

```
/*
    *情况一：target在数组两端之外，返回{-1，-1}
    *情况三：target在数组范围内，且数组中存在target，返回左右边界
    *情况二：target在数组范围内，且数组中不存在target，返回{-1.-1}
*/

class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        int leftBorder = getLeftBorder(nums,target);
        int rightBorder = getRightBorder(nums,target);
        if(leftBorder==-2 || rightBorder ==-2) { // 这种情况表明目标值在两端之外
            return {-1,-1};
        } else if((rightBorder-leftBorder)>1){
            return {leftBorder+1,rightBorder-1};
        } else{
            return {-1,-1};
        }
        // 下面的写法错误
        // if((rightBorder-leftBorder)>1) {
        //     return {leftBorder+1,rightBorder-1};
        // }
        // return {-1,-1};
    }

private:
    int getRightBorder(vector<int>&nums,int target){
        int rightBorder = -2;
        int left = 0;
        int right = nums.size()-1;
        while(left<=right){
            int middle = left + (right-left)/2;
            if(nums[middle]>target){
                right = middle-1;
            } else { // 当遇到nums[middle] == target后，从左向右逼近就不可能有nums[middle] < target，除非==，否则下面的语句不执行
                left = middle+1;
                rightBorder = left; // rightBorder比真实的右边界大1
            }
        }
        return rightBorder;
    }

    int getLeftBorder(vector<int>&nums,int target){
        int leftBorder = -2;
        int left = 0;
        int right = nums.size()-1;
        while(left<=right){
            int middle = left + (right-left)/2;
            if(nums[middle]>=target){ // 遇到nums[middle]==target后，从右向左逼近就不会有nums[middle]>target，除非==，否则下面的语句块不执行，leftBoard不会更新
                right = middle-1;
```

```
                leftBorder = right; // leftBorder比真实左边界小1
            } else {
                left = middle+1;
            }

        }
```

69.x的平方根

```
class Solution {
public:
    int mySqrt(int x) {
        int left = 0;
        int right = x;
        int result = 0;
        while(left<=right){

            int middle = left + (right - left)/2;

            if((long long)middle*middle <= x){
                result = middle; // 注意这里可不是在找右边界
                left = middle+1;
            } else {
                right = middle -1;
            }

        }
        return result;

    }
};
```

367.判断一个数是否是完全平方数（可以写成某个整数的平方）

```
class Solution {
public:
    bool isPerfectSquare(int num) { // 1 的完全平方数就是1，选择[]的写法
        int left = 0;
```

Type *Markdown* and LaTeX: α^2