# 题目

给出一个按非递减顺序排列的数组 nums，和一个目标数值 target。假如数组 nums 中绝大多数元素的数值都等于 target，则返回 True，否则请返回 False。

所谓占绝大多数，是指在长度为 N 的数组中出现必须 超过 N/2 次。

示例 1：

输入：nums = [2,4,5,5,5,5,5,6,6], target = 5

输出：true

解释：

数字 5 出现了 5 次，而数组的长度为 9。

所以，5 在数组中占绝大多数，因为 5 次 > 9/2。

示例 2：

输入：nums = [10,100,101,101], target = 101

输出：false

解释：

数字 101 出现了 2 次，而数组的长度是 4。

所以，101 不是 数组占绝大多数的元素，因为 2 次 = 4/2。

提示：

1 <= nums.length <= 1000

1 <= nums[i] <= 10^9

1 <= target <= 10^9

# 分析

## 方法一：遍历

思路：

我们先不考虑其他的条件，本题只需要我们判断目标数值 target 在数组中出现的次数是否超过N/2次。那么只需要遍历数组，记录目标出现的次数，最后和数组长度的一半比较即可。

代码

func isMajorityElement(nums []int, target int) bool {

count := 0

for i := 0; i < len(nums); i++ {

if nums[i] == target {

count++

}

}

return count > len(nums)/2

}

复杂度分析

时间复杂度：O(n)，遍历一次数组。其中n为数组nums的长度。

空间复杂度：O(1)，没有使用额外的空间。

C++代码：

class Solution {

public:

bool isMajorityElement(vector<int>& nums, int target) {

const int S = nums.size();

const int half = S / 2;

int count = 0;

for (int i = 0; i < S && count <= half; count += (nums[i++] == target));

return count > half;

}

};

## 方法二：遍历直到目标数字

思路

方法一没有考虑任何前置条件，但是题目告诉我们这是一个非递减顺序排列的数组nums，那么当发现nums[i] > target的时候就可以退出遍历直接比较了。

代码

func isMajorityElement(nums []int, target int) bool {

count := 0

for i := 0; i < len(nums); i++ {

if nums[i] == target {

count++

} else if nums[i] > target {

break

}

}

return count > len(nums)/2

}

复杂度分析

时间复杂度：O(n)，最慢遍历一次数组。其中n为数组nums的长度。

空间复杂度：O(1)，没有使用额外的空间。

## 方法三：双指针

思路

既然数组有序，那么我们还可以通过双指针的方法找到目标数字的左右下标，然后通过下标计算长度。

代码

func isMajorityElement(nums []int, target int) bool {

if len(nums) == 1 {

return nums[0] == target

}

left, right := 0, len(nums) - 1

for left < right {

if nums[left] < target {

left++

} else if nums[left] > target {

return false

}

if nums[right] > target {

right--

} else if nums[right] < target {

return false

}

if nums[left] == target && nums[right] == target {

break

}

}

return right - left + 1 > len(nums) / 2

}

复杂度分析

时间复杂度：O(n)，最慢遍历一次数组。其中n为数组nums的长度。

空间复杂度：O(1)，没有使用额外的空间。

## 方法四：两次二分查找

思路

查找有序数组中的数字最快的方法还是二分查找，在方法三的基础上，我们使用二分查找求目标数字的左右下标。

二分查找目标数字的左右下标的相关算法请看34。在排序数组中查找元素的第一个和最后一个位置 。

代码

func isMajorityElement(nums []int, target int) bool {

left, right := binarySearchLeft(nums, target), binarySearchRight(nums, target)

return left != -1 && right != -1 && right - left + 1 > len(nums)/2

}

func binarySearchLeft(nums []int, target int) int {

l, r := 0, len(nums) - 1

for l <= r {

mid := l + (r - l)/2

if nums[mid] >= target {

r = mid - 1

} else {

l = mid + 1

}

}

if l < len(nums) && nums[l] == target {

return l

}

return -1

}

func binarySearchRight(nums []int, target int) int {

l, r := 0, len(nums) - 1

for l <= r {

mid := l + (r - l)/2

if nums[mid] <= target {

l = mid + 1

} else {

r = mid - 1

}

}

if r >= 0 && nums[r] == target {

return r

}

return r - 1

}

复杂度分析

时间复杂度：O(logn)。由于二分查找每次将搜索区间大约划分为两等分，所以时间复杂度为O(logn)。二分查找的过程被调用了两次，所以总的时间复杂度是对数级别的。其中n为数组nums的长度。

空间复杂度：O(1)，没有使用额外的空间。

## 方法五：一次二分查找

思路

在方法四的基础上，我们还可以继续优化时间复杂度，只需要使用一次二分查找。找到边界下标，再判断边界下标加上数组一半的下标的数字是否等于目标数字。

算法

使用二分查找找到目标数字的最左边的下标left。

判断left + len(nums)/2 下标是否等于target，如果等于，则说明长度大于一半。

需要注意判断相加后的下标是否越界。

代码

func isMajorityElement(nums []int, target int) bool {

left := binarySearchLeft(nums, target)

return left != -1 && left + len(nums)/2 < len(nums) && nums[left + len(nums)/2] == target

}

func binarySearchLeft(nums []int, target int) int {

l, r := 0, len(nums) - 1

for l <= r {

mid := l + (r - l)/2

if nums[mid] >= target {

r = mid - 1

} else {

l = mid + 1

}

}

if l < len(nums) && nums[l] == target {

return l

}

return -1

}

复杂度分析

时间复杂度：O(logn)。由于二分查找每次将搜索区间大约划分为两等分，所以时间复杂度为O(logn)。其中n为数组nums的长度。

空间复杂度：O(1)，没有使用额外的空间。

解题思路（C++）

如果nums中存在绝大多数元素，那么必然是nums[(nums.size()-1)/2]，因此首先判断target是否等于nums[(nums.size()-1)/2]，如果不等于，则直接返回false

在nums中使用二分搜索找到target第一次出现的位置，及最后一次出现的位置，都是套模板，写不出来的需要自己整理一下各种二分查找模板了。

根据首次出现的位置和最后一次出现的位置来判断target的数目是都大于nums.size()/2;

代码

class Solution {

public:

bool isMajorityElement(vector<int>& nums, int target) {

int val = nums[(nums.size()-1)/2];

if (val != target) return false;

// 找到target首次出现的位置

int left = 0, right = nums.size()-1;

while (left < right) {

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

if (nums[mid] >= val) {

right = mid;

} else {

left = mid + 1;

}

}

int leftresult = left;

// 找到target最后一次出现的位置

left = 0, right = nums.size();

while (left < right) {

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

if (nums[mid] > val) {

right = mid;

} else {

left = mid+1;

}

}

int rightresult = left - 1;

return rightresult - leftresult + 1 > nums.size() / 2;

}

};

思路和心得：

1、本来是到简单题，直接遍历就行了。数据量也不大

纯粹就是想看看二分法

2、纯粹就是想手写一下二分查找的库函数

c++对应lower\_bound, upper\_lound

python3对应bisect.bisect\_left, bisect.bisect\_right

class Solution {

public:

bool isMajorityElement(vector<int>& nums, int target)

{

int n = nums.size();

//int L = lower\_bound(nums.begin(), nums.end(), target) - nums.begin(); //第一次出现的位置 ≥

int L = binary\_search\_left(nums, 0, n, target); //手写lower\_bound()

//int R = upper\_bound(nums.begin(), nums.end(), target) - nums.begin(); //第一个不是的位置 ＞

int R = binary\_search\_right(nums, 0, n, target); //收写upper\_bound()

int cur\_len = R - L;

return (cur\_len > n / 2 );

}

int binary\_search\_left(vector<int> & nums, int L, int R, int target)

{

while (L < R)

{

int mid = (L + R) >> 1;

if (nums[mid] >= target) //寻找符合条件的最左端

R = mid;

else

L = mid + 1;

}

return L;

}

int binary\_search\_right(vector<int> & nums, int L, int R, int target)

{

while (L < R)

{

int mid = (L + R) >> 1;

if (nums[mid] > target) //寻找符合条件的最左端

R = mid;

else

L = mid + 1;

}

return L;

}

};