二分搜索 Binary Search

无隅

从一个简单问题说起

- 问题：给定一个排序并不存在重复元素的数组：[1，2，5，7，8，9，13]，查找8的位置
- 直观想法：遍历整个数组，找到与给定值相同的元素，返回下标
- 时间复杂度为O(n)

二分搜索

- 二分搜索将目标值与数组的中间元素进行比较
- 如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。如果在某一步骤数组为空，则代表找不到。
- 这种搜索算法每一次比较都使搜索范围缩小一半。
- 在排序数组中搜索的最快方法

二分搜索模版

```java
public int binarySearch(int[] nums, int target) {
    if (nums == null || nums.length == 0) {
        return -1;
    }

    int start = 0;
    int end = nums.length - 1;

    while (start + 1 < end) {
        int mid = start + (end - start) / 2;
        if (target < nums[mid]) {
            end = mid;
        }
        else if (target > nums[mid]) {
            start = mid;
        }
        else {
            end = mid;
        }
    }

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

    return -1;
}
```

二分搜索代码要点

- 对输入做异常处理：数组为空或者数组长度为0。
- int mid = start + (end - start) / 2 这种表示方法可以防止两个整型值相加时溢出。
- Recursion or While-Loop：使用迭代而不是递归进行二分查找，因为工程中递归写法存在潜在溢出的可能
- while循环终止条件：while终止条件应为start + 1 < end而不是start <= end，start == end时可能出现死循环，即循环终止条件是相邻或相交元素时退出。配合while终止条件start + 1 < end（相邻即退出）的赋值语句mid永远没有+1或者-1，这样不会死循环。
- 迭代终止时target应为start或者end中的一个。循环终止条件有两个，具体应看是找第一个还是最后一个而定。

为什么不写成start <= end

- input： [3,4,5,8,8,8,8,10,13,14]， 找到第一个“8”出现的位置
- output： 4
- expect： 3

面试中是否使用递归的tips

- 面试官是否要求了不使用递归 （如果你不确定，就向面试官询问）
- 不用递归会不会实现起来很复杂
- 递归的深度是否会很深，会不会造成溢出
- 记住：不要自己下判断，要跟面试官讨论！

二分搜索的时间复杂度

- 二分搜索 $T(n) = T\left(\frac{n}{2}\right) + m$
- $a = 1, b = 2, c = 0,\ log_b a = c = 0$
- 符合主定理的第二种情况 -> $T(n) = n^c logn = logn$

- $T(n) = O(logn)$

二分搜索高频题目

第一个错误的版本

你是产品经理，目前正在领导一个团队开发一个新产品。不幸的是，您的产品的最新版本没有通过质量检查。由于每个版本都是基于之前的版本开发的，所以错误版本之后的所有版本都是不好的。

假设你有 n 个版本 [1, 2, ..., n]，你想找出第一个错误的版本，导致下面所有的错误。

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

https://leetcode-cn.com/problems/first-bad-version/description/

```java
public int firstBadVersion(int n) {
    if (n == 0) {
        return -1;
    }

    int start = 1;
    int end = n;
    while (start + 1 < end) {
        int mid = start + (end - start) / 2;
        if (isBadVersion(mid) == false) {
            start = mid;
        } else {
            end = mid;
        }
    }

    if (isBadVersion(start) == true) {
        return start;
    }

    if (isBadVersion(end) == true) {
        return end;
    }

    return -1;
}
```

搜索插入位置

给定一个排序数组和一个目标值，如果在数组中找到目标值则返回索引。
如果没有，返回到它将会被按顺序插入的位置。
你可以假设在数组中无重复元素。

case 1:
输入: [1,3,5,6], 5 输出: 2
case 2:
输入: [1,3,5,6], 2 输出: 1
case 3:
输入: [1,3,5,6], 7 输出: 4

https://leetcode.com/problems/search-insert-position/description/

```java
public int searchInsert(int[] nums, int target) {
    if (nums == null || nums.length == 0) {
        return -1;
    }

    int start = 0;
    int end = nums.length - 1;
    while (start + 1 < end) {
        int mid = start + (end -start) / 2;
        if (target > nums[mid]) {
            start = mid;
        } else {
            end = mid;
        }
    }

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

    if (target < nums[0]) {
        return 0;
    }
    if (nums[end] < target){
        return end + 1;
    }
    return start + 1;
}
```

搜索二维矩阵

编写一个高效的算法来搜索 m x n 矩阵中的一个目标值。
该矩阵具有以下特性：
每行中的整数从左到右排序。
每行的第一个整数大于前一行的最后一个整数。

例如，
以下矩阵：

```
[
  [1, 3, 5, 7],
  [10, 11, 16, 20],
  [23, 30, 34, 50]
 ]
```

给定目标值= 3，返回 true。
https://leetcode-cn.com/problems/search-a-2d-matrix/description/

```java
public boolean searchMatrix(int[][] matrix, int target) {
    if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {
        return false;
    }
    int row = matrix.length;
    int col = matrix[0].length;
    int start = 0;
    int end = row * col - 1;
    while (start + 1 < end) {
        int mid = start + (end - start) / 2;
        int x = mid / col ;
        int y = mid % col ;
        if (matrix[x][y] == target) {
            end = mid;
        }
        else if (matrix[x][y] < target) {
            start = mid;
        }
        else {
            end = mid;
        }
    }
    if (matrix[start / col][start % col] == target) {
        return true;
    }
    if (matrix[end / col][end % col] == target) {
        return true;
    }

    return false;
}
```

搜索二维矩阵 II

编写一个高效的算法来搜索 m x n 矩阵中的一个目标值。
该矩阵具有以下特性：
每行的元素从左到右升序排列。
每列的元素从上到下升序排列。
例如，
考虑下面的矩阵：

```
[ [1, 4, 7, 11, 15],
  [2, 5, 8, 12, 19],
  [3, 6, 9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30] ]
```

给定目标值 target = 5, 返回 true。
给定目标值 target = 20, 返回 false。
https://leetcode-cn.com/problems/search-in-rotated-sorted-array-ii/description/

```java
public boolean searchMatrix(int[][] matrix, int target) {
    if (matrix == null || matrix.length == 0) {
        return false;
    }

    if (matrix[0] == null || matrix[0].length == 0) {
        return false;
    }

    int rowBegin = 0;
    int rowEnd = matrix.length - 1;
    int colBegin = 0;
    int colEnd = matrix[0].length - 1;


    while(colEnd >= colBegin && rowBegin <= rowEnd) {
        int temp = matrix[rowBegin][colEnd];
        if(target == temp) {
            return true;
        }
        else if (target < temp) {
            colEnd--;
        }
        else if (target > temp) {
            rowBegin++;
        }

    }

    return false;
}
```

x 的平方根

实现 int sqrt(int x) 函数。

计算并返回 x 的平方根。

x 保证是一个非负整数。

https://leetcode-cn.com/problems/sqrtx/description/

```java
public int mySqrt(int x) {
    if (x == 0) {
        return 0;
    }
    if (x < 0) {
        return -1;
    }
    long start = 1;
    long end = x ;
    while (start + 1 < end) {
        long mid = start + (end - start) / 2;
        if (mid * mid == x ) {
            return (int) mid;
        }
        else if (mid * mid < x){
            start = mid;
        }
        else {
            end = mid;
        }
    }
    if (end * end <= x){
        return (int) end;
    }
    return (int) start;
}
```

```java
public int mySqrt(int x) {
    if (x == 0) {
        return 0;
    }
    if (x < 0) {
        return -1;
    }
    int start = 1;
    int end = x ;
    while (start + 1 < end) {
        int mid = start + (end - start) / 2;
        if (mid == x / mid) {
            return mid;
        }
        else if (mid < x / mid ){
            start = mid;
        }
        else {
            end = mid;
        }
    }
    if (end <= x / end){
        return end;
    }
    return start;
}
```

搜索旋转排序数组

假设按照升序排序的数组在预先未知的某个关键点上旋转。

（即 0 1 2 4 5 6 7 将变成 4 5 6 7 0 1 2）。

给你一个目标值来搜索，如果数组中存在这个数则返回它的索引，否则返回 -1。

你可以假设数组中不存在重复。

https://leetcode-cn.com/problems/search-in-rotated-sorted-array/description/

```java
public int search(int[] nums, int target) {
    if (nums.length == 0 || nums == null) {
        return -1;
    }

    int start = 0;
    int end = nums.length - 1;

    while (start + 1 < end) {
        int mid = start + (end - start) / 2;
        if (nums[mid] == target) {
            end = mid;
        } else if (nums[mid] < nums[end]) {
            if (nums[mid] <= target && target <= nums[end]) {
                start = mid;
            } else {
                end = mid;
            }
        } else {
            if (nums[mid] >= target && target >= nums[start]) {
                end = mid;
            } else {
                start = mid;
            }
        }
    }

    if (nums[start] == target) {
        return start;
    }
    if (nums[end] == target) {
        return end;
    }
    return -1;
}
```

总结

- 理解二分法的三个层次：

1. 头尾指针，取中点，判断往哪儿走
2. 寻找满足某个条件的第一个或是最后一个位置
3. 保留剩下来一定有解的那一半

- 二分法模板的四点要素

  start + 1 < end

  start + (end - start) / 2

  nums[mid] ==, <, >

  nums[start] nums[end] 与target关系