二分查找（面试必备）

2017 年 03 月 15 日 • 技术

在计算机科学中，二分搜索（binary search），也称折半搜索（half-interval search）、对数搜索（logarithmic search），是一种在**有序数组**中查找某一特定元素的搜索算法。搜索过程从数组的中间元素开始，如果中间元素正好是要查找的元素，则搜索过程结束；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。

问题 1

给定一个有序的数组，查找 value 是否在数组中，不存在返回 - 1。

例如：{1, 2, 3, 4, 5} 找 3，返回下标 2（下标从 0 开始计算）。

```
/* 注意：题目保证数组不为空，且 n 大于等于 1 ，以下问题默认相同 */
int BinarySearch(int array[], int n, int value)
{
    int left = 0;
    int right = n - 1;
    // 如果这里是 int right = n 的话，那么下面有两处地方需要修改，以保证一一对应：
    // 1、下面循环的条件则是 while(left < right)
    // 2、循环内当 array[middle] > value 的时候，right = middle

    while (left <= right)  // 循环条件，适时而变
    {
        int middle = left + ((right - left) >> 1);  // 防止溢出，移位也更高效。同时，每次循环都需要更新。
        if (array[middle] > value)
            right = middle - 1;  // right 赋值，适时而变
        else if (array[middle] < value)
            left = middle + 1;
        else
            return middle;
        // 可能会有读者认为刚开始时就要判断相等，但毕竟数组中不相等的情况更多
        // 如果每次循环都判断一下是否相等，将耗费时间
    }

    return -1;
}
```

问题 2

给定一个有序的数组，查找第一个等于 value 的下标，找不到返回 - 1。

例如：{1, 2, 2, 2, 4} 找 2，返回下标 1（下标从 0 开始计算）。

```
int BinarySearch(int array[], int n, int value)
{
    int left = 0;
    int right = n - 1;

    while (left <= right)
    {
        int middle = left + ((right - left) >> 1);

        if (array[middle] >= value)  // 因为是找到最小的等值下标，所以等号放在这里
            right = middle - 1;
        else
            left = middle + 1;
    }

    if (left < n && array[left] == value)
        return left;

    return -1;
}
```

如果问题改为 "查找 value 最后一个等于 value 的下标" 呢？只需改动两个位置：

1. `if (array[middle] >= value)` 中的等号去掉；

2.
```
if (left < n && array[left] == value)
    return left;
```

改为

```
if (right >= 0 && array[right] == value)
    return right;
```

问题 3

给定一个有序的数组，查找第一个大于等于 value 的下标，都比 value 小则返回 - 1。

也就是说如果有等于 value 的返回第一个等于 value 的下标，如果没有则返回第一个大于 value 的下标。

```
int BinarySearch(int array[], int n, int value)
{
    int left = 0;
    int right = n - 1;

    while (left <= right)
    {
        int middle = left + ((right - left) >> 1);

        if (array[middle] >= value)
```

```
            right = middle - 1;
        else
            left = middle + 1;
    }

    return (left < n) ? left : -1;
}
```

如果问题改为 "查找最后一个小于等于 value 的下标" 呢？只需改动两个位置：

1. `if (array[middle] >= value)` 的等号去掉；
2. `return (left < n) ? left : -1` 改为 `return (right >= 0) ? right : -1` 。

问题 4

给定一个轮转后的有序数组（所谓转轮有序数组，比如：{2, 3, 4, 5, 1}，{ 5, 1, 2, 3, 4 }），查找 value 是否在数组中，不存在返回 - 1。

```
int BinarySearch(int array[], int n, int value)
{
    int left = 0;
    int right = n - 1;

    while (left <= right)
    {
        int middle = left + ((right - left) >> 1);

        if (value < array[middle])
        {
            if (array[middle] < array[right])
                right = middle - 1;
            else
            {
                if (value < array[left])
                    left = middle + 1;
                else
                    right = middle - 1;
            }
        }
        else if (value > array[middle])
        {
            if (array[middle] > array[left])
                left = middle + 1;
            else
            {
                if (value > array[right])
                    right = middle - 1;
                else
                    left = middle + 1;
            }
        }
        else
```

```
            return middle;
        }

        return -1;
    }
```

理解上面的代码很简单，只需从三个方面考虑：

{1, 2, 3, 4, 5}，

{2, 3, 4, 5, 1}，

{5, 1, 2, 3, 4}。

总结

二分算法所操作的区间，是左闭右开，还是左闭右闭，需要在循环体跳出判断中，以及每次修改 left,，right 区间值这两个地方保持一致，否则就可能出错。

另外，我发现一个小技巧（个人观点，正确性未经考证，仅作参考）：

当我们写好一个二分程序的时候，总希望找到一些数据来考证我们的程序的正确性，但差强人意的是，我们总会遗漏某个测试数据，从而导致程序依旧存在 bug。而我想说的是，我的这个小技巧可以把针对二分的成千上万所有的测试数据都压缩成与之等价的屈指可数的几个测试数据。

我们知道二分的思想就是每次取一半，想象一下，不管给我们的数组有多长，每次取一半，最终都会被压缩成长度为 1 的数组，然后在这个长度为 1 的数组里判断并返回，所以我们可以直接用长度为 1 的数组来测试程序。以上述的 "问题 3：给定一个有序的数组，查找第一个大于等于 value 的下标，都比 value 小则返回 - 1" 为例。

- 若找不到。例如数组为 {0}，value = 1，则 left = 1，right = 0，left 越界；
- 若可以找到；
    - 找到等于 value 的。例如数组为 {0}，value = 0，则 left = 0，right = -1，right 越界；
    - 找到大于 value 的。例如数组为 {0}，value = -1，则 left = 0，right = -1，right 越界；

对比程序最后的返回语句 `return (left < n) ? left : -1` ，代码正确。

参考文献

- 维基百科. 二分搜索算法.
- GitHub. 有序数组的查找.
- 二分查找.

标签: 二分查找

返回文章列表 文章二维码 打赏