变形二分总结

原创 张忠诚 LOA算法学习笔记 2021-02-18 20:22

二分查找是一种非常简单的快速查找算法，常规的二分查找思路非常简单，就是待查找值与有序数组中间值做比较，如果待查找值小于中间值就在左子数组寻找，如果大于中间值就在右子数组寻找，否则中间值就是要找的元素。虽然常规二分比较简单，但是二分查找却存在非常多的变形问题，比如DC作业的第4题以及考试的第1题都可算是一种变形二分。二分变形问题有很多，下面仅对几种典型的二分变形问题做个总结。

01 常规的二分查找

常规的二分查找思路比较简单且查找性能非常高效，但他却有一定的适用范围，比如他依赖的数据结构是顺序表而不是链表，查找的数据必须是有序的，且从数据量上来看，太多太少的数据量都不适合用二分，数据太少比如是个位数时，二分查找和顺序遍历的耗时差不太多，我们没有必要使用二分查找。而数据量太大的时候，由于二分查找依赖的数据结构是顺序表，顺序表要求内存空间连续，所以由于内存的限制也不适合使用二分。下面是常规二分查找的代码，尤其要注意mid取值的溢出问题。

```
int BinarySearch(int *a, int n, intsearchValue) {
    intlow = 0;
    inthigh = n - 1;
    while(low <= high) {
       //int mid = (low + high) / 2;//可能会导致溢出
       int mid = low + ((high - low) >> 1);//选用这种写法避免溢出
       if (a[mid] == searchValue) {
           return mid;
        }else if (a[mid] < searchValue) {
           low = mid + 1;
        }else {
           high = mid - 1;
        }
    }
   return -1;
}
```

02 查找第一个等于给定值的元素

如果有序数组中存在重复元素且给定值为某一重复元素的话，我们常规的二分查找只能够保证找到该元素但却不能保证找到的是第一个等于该值的元素，这就要求我们对上面的二分查找算法做一定的变形。下面先给出相应的代码，在对代码做解释的同时阐述思路。

```
int BinarySearch(int *a, int n, int searchValue) {
    int low = 0;
    int high = n - 1;
    while (low <= high) {
        int mid = low + ((high - low) >> 1);
```

```
        if (a[mid] < searchValue) {
            low = mid + 1;
        } else if(a[mid] > searchValue) {
            high = mid - 1;
        } else {
            if (mid == 0 || a[mid - 1] != searchValue) {
                return mid;
            } else {
                high = mid - 1;
            }
        }
    }
    return -1;
}
```

查找第一个等于给定值的元素就意味着当我们找到等于给定值的元素时不一定能够直接返回，需要判断他左边的元素是不是还等于给定值，如果等于就需要去左边的子区间继续寻找。根据代码我们也能看出，当我们找到等于给定值的元素时，只有当其是数组的第一个元素或其左边的元素不等于给定值时我们才直接返回，否则我们就令high = mid-1继续去左边区间寻找。

03 查找最后一个等于给定值的元素

这个问题跟上面的问题是对称的，根据上面问题的分析我们很容易想出本问题的思路，即当我们找到与给定值元素相等的元素时，首先需要看其是不是数组的最后一个元素，如果是那么直接返回，如果不是还需要看其右边的元素跟其是不是相等，如果不想等则直接返回，否则就需要继续去右边区间寻找。以下是该问题相应的代码，我们只需要变动else部分即可。

```
int BinarySearch(int *a, int n, int searchValue) {
    int low = 0;
    int high = n - 1;
    while (low <= high) {
        int mid = low + ((high - low) >> 1);
        if (a[mid] < searchValue) {
            low = mid + 1;
        } else if(a[mid] > searchValue) {
            high = mid - 1;
        } else {
            if (mid == n - 1 || a[mid + 1] != searchValue) {
                return mid;
            } else {
                low = mid + 1;
            }
        }
    }
    return -1;
}
```

04 查找第一个大于等于给定值的元素

前面的三种二分查找，无论怎么变化，我们总是在找相等，而这个变形问题我们除了找相等外，又多了一种选择，比如有这么一个序列，1，2，3，5，6，如果给定值是4，那我们返回的结果就应该是5。显然当中间元素小于给定值的时候，我们就去它的右半区间寻找，否则就意味着我们找到了一个符合部分要求(大于等于给定值)的元素，这时候我们就需要判断它是不是第一个，如果它是数组的首元素显然是，否则就需要继续判断它左边的元素是不是符合部分要求，如果不符合就说明它是第一个，否则就说明第一个还在该元素的左边，需要去左边子数组继续寻找。以下是该问题相应的代码，与前面的代码类似甚至还更为简洁。

```
int BinarySearch(int *a, int n, int searchValue) {
    int low = 0;
    int high = n - 1;
    while (low <= high) {
        int mid = low + ((high - low) >> 1);
        if (a[mid] < searchValue) {
            low = mid + 1;
        } else {
            if (mid == 0 || a[mid - 1] < searchValue) {
                return mid;
            } else {
                high = mid - 1;
            }
        }
    }
    return -1;
}
```

05 查找最后一个小于等于给定值的元素

本问题与上一问题是对称的，根据上一问题的思路我们很容易想出本问题的思路。即当我们找到一个小于等于给定值的元素时，需要先判断它是不是数组的尾元素，如果是则直接返回，否则需要进一步判断它右边的元素是不是仍小于等于给定值，如果大于则直接返回，否则就需要去其右边的区间继续寻找，以下是该问题对应的代码。

```
int BinarySearch(int *a, int n, int searchValue) {
    int low = 0;
    int high = n - 1;
    while (low <= high) {
        int mid = low + ((high - low) >> 1);
        if (a[mid] > searchValue) {
            high = mid - 1;
        } else {
            if (mid == n - 1 || a[mid + 1] > searchValue) {
                return mid;
            } else {
```

```
                low = mid + 1;
            }
        }
    }
    return -1;
}

```

06 总结

以上就是四种比较常见的查找值的二分变形，有些二分变形可能就是这几种的变种或者组合，比如DC作业的第四题，让我们查找给定值所在的区间，其实就是第一种变形和第二种变形的组合。二分查找除了查找值这种较为直观的应用外，有些最大化最小值，最小化最大值之类的问题也可以通过二分查找解决，大家感兴趣的话可以看一看leetcode上410分割数组的最大值和1552两球之间的磁力这两道题目。