

二分查找（下）：如何快速定位IP对应的省份地址？

假设我们有12万条这样的 IP 区间与归属地的对应关系，如何快速定位出一个 IP 地址的归属地呢？

二分查找的变形问题

四种常见的二分查找变形问题

1. 查找第一个值等于给定值的元素
2. 查找最后一个值等于给定值的元素
3. 查找第一个大于等于给定值的元素
4. 查找最后一个小于等于给定值的元素

(以数据是从小到大排列为前提)

变体一：查找第一个值等于给定值的元素

上一节中的二分查找是最简单的一种，即有序数据集中不存在重复的数据；当有序数据集中存在重复的数据，我们希望找到第一个值等于给定的数据。找到第一个为8的元素。



代码实现：

```
/**
 * 寻找第一个等于给定值的元素
 * @param a
 * @param value
 * @return
 */
public int fristItem(int[] a,int value){
    int low = 0;
    int high = a.length - 1;
    while(low <= high){
        int mid = low + ((high - low) >> 1);
        if(a[mid] > value){
            high = mid - 1;
        }else if(a[mid] < value){
            low = mid + 1;
        }else {
            if(mid == 0 || a[mid - 1] != value) return mid;
            else high = mid - 1;
        }
    }
}
```

```

    }
}
return -1;
}

```

$a[mid]$ 跟要查找的 $value$ 的大小关系有三种情况：大于、小于、等于。对于 $a[mid] > value$ 的情况，我们需要更新 $high = mid - 1$ ；对于 $a[mid] < value$ 的情况，我们需要更新 $low = mid + 1$ ；当 $a[mid] = value$ 的时候，如果 mid 等于 0，那这个元素已经是数组的第一个元素，那它肯定是要找的；如果 mid 不等于 0，但 $a[mid]$ 的前一个元素 $a[mid - 1]$ 不等于 $value$ ，那也说明 $a[mid]$ 就是我们要找的第一个值等于给定值的元素。

变形二：查找最后一个值等于给定值的元素

```

/**
 * 查找最后一个等于给定值的函数
 * @param a
 * @param value
 * @return
 */
public int lastItem(int[] a, int value){
    int low = 0;
    int high = a.length - 1;
    while (low <= high){
        int mid = low + ((high - low) >> 1);
        if(a[mid] > value){
            high = mid - 1;
        }else if(a[mid] < value){
            low = mid + 1;
        }else {
            if(mid == a.length - 1 || a[mid + 1] != value) {
                return mid;
            }else {
                low = mid + 1;
            }
        }
    }
    return -1;
}

```

最后给定值的元素。

变形三：查找第一个大于等于给定值的元素

实现代码：

```

/**
 * 第一个大于等于给定值的元素
 * @param a

```

```
* @param value
* @return
*/
public int fristGEItem(int[] a,int value){
    int low = 0;
    int high = a.length - 1;
    while(low <= high){
        int mid = low + ((high - low) >> 1);
        if(a[mid] < value){
            low = mid + 1;
        }else {
            if(mid == 0 || a[mid - 1] < value){
                return mid;
            }else {
                high = mid - 1;
            }
        }
    }
    return -1;
}
```

变体四：查找最后一个小于等于给定值的元素

```
/**
 * 最后一个小于等于给定值的元素
 * @param a
 * @param value
 * @return
 */
public int lastLEItem(int[] a,int value){
    int low = 0;
    int high = a.length - 1;
    while(low <= high){
        int mid = low + ((high - low) >> 1);
        if(a[mid] > value){
            high = mid - 1;
        }else {
            if(mid == a.length - 1 || a[mid + 1] > value){
                return mid;
            }else {
                low = mid + 1;
            }
        }
    }
    return -1;
}
```

IP地址可以转化为32位的整形数，所以我们可以将起始地址，按照对应的整形值的大小关系从小到大进行排序。

这样IP地址查找的问题就变成第四种变形问题，查找最后一个小于等于某个给定值的元素。

当我们要查询某个 IP 归属地时，我们可以先通过二分查找，找到最后一个起始 IP 小于等于这个 IP 的 IP 区间，然后，检查这个 IP 是否在这个 IP 区间内，如果在，我们就取出对应的归属地显示；如果不在，就返回未查找

课后思考

如果有序数组是一个循环有序数组，比如 4, 5, 6, 1, 2, 3。针对这种情况，如何实现一个求“值等于给定值”的二分查找算法呢？

```
public int search1(int[] nums, int target){
    return search1(nums,0,nums.length - 1, target);
}

public int search1(int[] nums,int low,int high,int target){
    if(low > high){
        return -1;
    }
    int mid = low + ((high - low) >> 2);
    if(nums[mid] == target){
        return mid;
    }
    if(nums[mid] < nums[high]){
        if(target <= nums[high] && target > nums[mid]){
            return search1(nums,mid + 1,high,target);
        }else {
            return search1(nums,low,mid - 1,target);
        }
    }else {
        if(target >= nums[low] && target < nums[mid]){
            return search1(nums,low,mid - 1,target);
        }else {
            return search1(nums,mid + 1,high,target);
        }
    }
}
```