二分搜索法（下）

在前面的课程中，我们学习了最基本的二分搜索模型。但是该模型存在一个Bug，我们来看下面的例子。

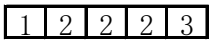
现在给定数组[1,2,2,2,3]，我们的目标数字为2，在运行了上节课的基本搜索后，我们发现结果是2，也就是数组中[1,2,2,2,3]标红的数字。

如果本题想要求最左边的2，或者最右边的2，那么原有的模型是无法处理的。今天我们就来学习如何处理这种情况。

简单思考一下，如果我们在函数第一次找到目标时，并不停止搜索，而是将边界搜索至当前的位置，继续向前搜索，是不是就能找到最终我们想要的结果了呢？我们先来尝试搜索位于最左侧的目标数字索引。

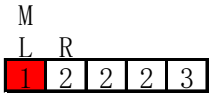
观察下面的图示：

最初时刻的数组

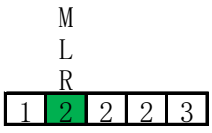


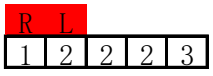
第一次搜索，我们的运气非常好，首次就找到了目标！

但是，我们找到目标后，并不停止，而是继续向左，即将R设置为M--；为什么这样设置？别忘了我们是在找最左侧的目标索引。结果如下图：



可以看到此时的M与L重合了，数组[M]的值小于目标数字，我们需要将左边界增加，即L=M++，如下图所示：





下一次的循环结果是R将继续减小变为1（为什么是R变？因为我们在搜索左边界，如上图所示），不再满足搜索条件，搜索终止。此时，我们返回L便是期望的正确答案：索引1。

原理明白之后，我们就可以开始编写代码了。

注意，对于最后的结果，我们需要对L进行校验，因为很有可能目标数据并不存在。

编码

```
//nums 待搜索数组
//target 目标数字
//length 数组长度
//return 目标所在索引位置
int BinarySearchLeft(int nums[], int target, int length) {
    int left = 0; //左边界
   //我们将长度进行了减1
    int right = length - 1;
    //开始循环搜索
   while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            //持续向左搜索
          right = mid - 1;
        }
            //小于目标，边界右移
       else if (nums[mid] < target) {
            left = mid + 1;
        }
            //大于目标，边界左移
       else if (nums[mid] > target) {
            right = mid - 1;
        }
    }
    //寻找的目标数字不存在于当前数组
   if (left >= length || nums[left] != target) {
        return -1;
    }
    return left;
}
```

同理可推，如果我们想计算右边界的目标值索引，只需要稍加改动即可，具体代码如下：

```
//nums 待搜索数组
//target 目标数字
//length 数组长度
//return 目标所在索引位置
int BinarySearchRight(int nums[], int target, int length) {
    int left = 0; //左边界
    //我们将长度进行了减1
    int right = length - 1;
    //开始循环搜索
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (nums[mid] == target) {
            //持续向右搜索,左边界增加
          left = mid + 1;
        }
            //小于目标，边界右移
       else if (nums[mid] < target) {
            left = mid + 1;
        }
            //大于目标，边界左移
       else if (nums[mid] > target) {
            right = mid - 1;
        }
    }
    if (right >= length || nums[right] != target) {
        return -1;
    }

    return right;
}
```

上面的红色代码处，就是我们最终修改的地方。怎么样，是不是非常的简单？