```
if (arr[mid] > arr[high]) {
        low = mid;
        continue;
}
while (low < mid) {
        if (arr[low] == arr[mid]) {
            low++;
        } else if (arr[low] < arr[mid]) {
                return arr[low];
        } else {
                high = mid;
                break;
        }
}
return Math.min(arr[low], arr[high]);</pre>
```

在有序旋转数组中找到一个数

【题目】

有序数组 arr 可能经过一次旋转处理,也可能没有,且 arr 可能存在重复的数。例如,有序数组[1,2,3,4,5,6,7],可以旋转处理成[4,5,6,7,1,2,3]等。给定一个可能旋转过的有序数组 arr,再给定一个数 num,返回 arr 中是否含有 num。

【难度】

尉★★☆☆

【解答】

为了方便描述,我们把没经过旋转前,有序数组 arr 最左边的数,在经过旋转之后所处的位置叫作断点。例如,题目例子里的数组,在旋转后断点在 1 所处的位置,也就是位置 4。如果一个数组没有经过旋转处理,断点在位置 0。

本书提供的方式做到了尽可能多地利用二分查找,但是最差情况下仍无法避免 O(N)的时间复杂度,以下是具体过程:

- 1. 用 low 和 high 变量表示 arr 上的一个范围,每次判断 num 是否在 arr[low..high]上, 初始时,low=0,high=arr.length-1,然后进入步骤 2。
 - 2. 如果 low>high, 直接进入步骤 5, 否则令变量 mid=(low+high)/2, 也就是二分的位

- 置。如果 arr[mid]==num,直接返回 true,否则进入步骤 3。
- 3. 此时 arr[mid]!=num。如果发现 arr[low]、arr[mid]、arr[high]三个值不都相等,直接进入步骤 4。如果发现三个值都相等,此时根本无法知道断点的位置在 mid 的哪一侧。例如: 7(low)···7(mid)···7(high),举一个极端的例子,如果这个数组中只有一个值为 num 的数,其他的数都是 7,那么 num 除了不在 low、mid、high 这三个位置以外,剩下的位置都是可能的。所以 num 也既可能在 mid 的左边,也可能在右边。所以进行这样的处理:
- 1) 只要 arr[low]等于 arr[mid], 就让 low 不断地向右移动 (low++), 如果在 low 移到 mid 的期间,都没有发现 arr[low]和 arr[mid]不等的情况,说明 num 只可能在 mid 的右侧,因为左侧全都扫过了,此时令 low=mid+1, high 不变,进入步骤 2。
- 2) 只要 arr[low]等于 arr[mid],就让 low 不断地向右移动(low++),如果期间一旦发现 arr[low]和 arr[mid]不等,说明在此时的 arr[low(递增后的)..mid..right]上是可以判断出断点位置的,则进入步骤 4。
- 4. 此时 arr[mid]!=num, 并且 arr[low]、arr[mid]、arr[high]三个值不都相等,那么是一定可以二分的,具体判断如下:

如果 arr[low]!=arr[mid],如何判断断点位置呢?分以下两种情况。

情况一: arr[mid]>arr[low], 断点一定在 mid 的右侧, 此时 arr[low..mid]上有序。

- 1) 如果 num>=arr[low]&&num<arr[mid],说明 num 只需要在 arr[low..mid]上寻找。这是因为如果 num==arr[low]&&num<arr[mid]。很显然,在 arr[low..mid]上能找到 num。如果 num>arr[low]&&num<arr[mid],则说明断点在右侧,假设断点在 mid 和 high 之间的 break 位置上,那么 arr[mid..break-1]上的值都大于或等于 arr[mid],也都大于 num,arr[break..high]上的值都小于或等于 arr[low],也都小于 num,所以整个 mid 的右侧都没有 num。综上所述,num 只需要在 arr[low..mid]上寻找,令 high=mid-1,进入步骤 2。
- 2) 若不满足条件 1),说明要么 num<arr[low],此时整个 arr[low..mid]上都大于 num。要么 num>arr[mid],此时整个 arr[low..mid]上都小于 num。无论是哪种, num 都只可能出现在 mid 的右侧,所以令 low=mid+1,进入步骤 2。
- 情况二:不满足情况一则断点一定在 mid 位置或在 mid 左侧,不管是哪一种, arr[mid..high]都一定是有序的。
- 1) 如果 num>arr[mid]&&num<=arr[high]与情况一的条件 1)相同的分析方式,令 low=mid+1,进入步骤 2。
 - 2) 若不满足条件 1),与情况一的条件 2)相同的分析方式,令 high=mid-1,进入步骤 2。 如果 arr[mid]!=arr[high],如何判断断点的位置呢?和 arr[low]!=arr[mid]时一样的分析

方式,这里不再详述。

5. 如果 low 在 high 的右边(low>high),说明 arr 中没有 num,返回 false。全部的过程请参看如下代码中的 isContains 方法。

```
public boolean isContains(int[] arr, int num) {
       int low = 0;
       int high = arr.length - 1;
       int mid = 0;
       while (low <= high) {
               mid = (low + high) / 2;
               if (arr[mid] == num) {
                      return true;
               if (arr[low] == arr[mid] && arr[mid] == arr[high]) {
                       while (low != mid && arr[low] == arr[mid]) {
                              low++;
                       if (low == mid) {
                              low = mid + 1;
                              continue;
               if (arr[low] != arr[mid]) {
                      if (arr[mid] > arr[low]) {
                              if (num >= arr[low] && num < arr[mid]) {</pre>
                                      high = mid - 1;
                              } else {
                                      low = mid + 1;
                       } else {
                              if (num > arr[mid] && num <= arr[high]) {
                                      low = mid + 1;
                              } else {
                                      high = mid - 1;
               } else {
                      if (arr[mid] < arr[high]) {
                              if (num > arr[mid] && num <= arr[high]) {
                                      low = mid + 1;
                              else {
                                      high = mid - 1;
                      } else {
                              if (num >= arr[low] && num < arr[mid]) {
                                     high = mid - 1;
                              } else {
                                      low = mid + 1;
```

```
}
return false;
```

数字的英文表达和中文表达

【题目】

给定一个 32 位整数 num, 写两个函数分别返回 num 的英文与中文表达字符串。

【举例】

num=319

英文表达字符串为: Three Hundred Nineteen

中文表达字符串为: 三百一十九

num=1014

英文表达字符串为: One Thousand, Fourteen

中文表达字符串为:一千零十四

num=-2147483648

英文表达字符串为: Negative, Two Billion, One Hundred Forty Seven Million, Four Hundred Eighty Three Thousand, Six Hundred Forty Eight

中文表达字符串为: 负二十一亿四千七百四十八万三千六百四十八

num=0

英文表达字符串为: Zero

中文表达字符串为:零

【难度】

校 ★★★☆

【解答】

本题的重点是考查面试者分析业务场景并实际解决问题的能力。本题实现的方式当然 是多种多样的,本书提供的方法仅是作者的实现,希望读者也能写出自己的实现。

英文表达的实现。英文的表达是以三个数为单位成一组的, 所以先要解决数字 1~999