12 分治：如何利用分治法完成数据查找？

前面课时中，我们学习了递归的思想，它是一种函数自我调用缩小问题规模的方法。这一课时我们继续学习另一种算法思维，分治法。

从定性的角度来看，分治法的核心思想就是“分而治之”。利用分而治之的思想，就可以把一个大规模、高难度的问题，分解为若干个小规模、低难度的小问题。随后，开发者将面对多个简单的问题，并很快地找到答案各个击破。在把这些简单问题解决好之后，我们通过把这些小问题的答案合并，就得到了原问题的答案。

分治法应用很广泛，很多高效率的算法都是以分治法作为其基础思想，例如排序算法中的快速排序和归并排序。

分治法是什么？

计算机求解问题所需的计算时间，与其涉及的数据规模强相关。简而言之，问题所涉及的数据规模越小，它所需的计算时间也越少；反之亦然。

我们来看一个例子：**在一个包含 n 个元素的无序数组中，要求按照从小到大的顺序打印其 n 个元素。**

假设我们采用 n 个元素之间的两两比较的计算方法，去得到从小到大的序列。分析如下：

当数据量 n = 1 时，不需任何计算，直接打印即可；

当数据量 n = 2 时 ，那需要做 1 次比较即可达成目标；

当数据量 n = 3 时，要对这 3 个元素进行两两比较，共计 3 次比较；

而当数据量 n = 10 时，问题就不那么容易处理了，我们需要 45 次比较（计算方式是 0.5*n(n-1) ）。

因此，要想通过上述方法直接解决一个规模较大的问题，其实是相当困难的。

基于此，**分治法的核心思想就是分而治之。**具体来说，它先将一个难以直接解决的大问题，分割成一些可以直接解决的小问题。如果分割后的问题仍然无法直接解决，那么就继续递归地分割，直到每个小问题都可解。

通常而言，这些子问题具备互相独立、形式相同的特点。这样，我们就可以采用同一种解法，递归地去解决这些子问题。最后，再将每个子问题的解合并，就得到了原问题的解。

分治法的价值

关于分治法，很多同学都有这样一个误区。那就是，当你的计算机性能还不错的时候，采用分治法相对于全局遍历一遍没有什么差别。

例如下面这个问题，**在 1000 个有序数字构成的数组 a 中，判断某个数字 c 是否出现过。**

第一种方法，全局遍历。 复杂度 O(n)。采用 for 循环，对 1000 个数字全部判断一遍。

第二种方法，采用二分查找。 复杂度 O(logn)。递归地判断 c 与 a 的中位数的大小关系，并不断缩小范围。

这两种方法，对时间的消耗几乎一样。那分治法的价值又是什么呢?

其实，在小数据规模上，分治法没有什么特殊价值。无非就是让代码显得更牛一些。只有在大数据集上，分治法的价值才能显现出来。

下面我们通过一个经典的案例带你感受分治法的价值。

假如有一张厚度为 1 毫米且足够柔软的纸，问将它对折多少次之后，厚度能达到地球到月球的距离?

这个问题看起来很异想天开。根据百度百科，地月平均距离是 384,403.9 千米，大约 39 万千米。粗看怎么也需要对折 1 万次吧? 但实际上，根据计算，我们只需要对折 39 次就够了。计算的过程是 2^39 = 549,755,813,888 = 55 万千米 >

39 万千米。那么，这个例子意味着什么呢？

我们回到前面讲到的在数组 a 中查找数字 c 的例子，如果数组 a 的大小拓展到 549,755,813,888 这个量级上，使用第二种的二分查找方法，仅仅需要 39 次判断，就能找到最终结果。相比暴力搜索的方法，性能优势高的不是一星半点！这也证明了，**复杂度为 O(logn) 相比复杂度为 O(n) 的算法，在大数据集合中性能有着爆发式的提高。**

分治法的使用方法

前面我们讲到分治法的核心思想是“分而治之”，当你需要采用分治法时，**一般原问题都需要具备以下几个特征：**

1. **难度在降低**，即原问题的解决难度，随着数据的规模的缩小而降低。这个特征绝大多数问题都是满足的。
2. **问题可分**，原问题可以分解为若干个规模较小的同类型问题。这是应用分治法的前提。
3. **解可合并**，利用所有子问题的解，可合并出原问题的解。这个特征很关键，能否利用分治法完全取决于这个特征。
4. **相互独立**，各个子问题之间相互独立，某个子问题的求解不会影响到另一个子问题。如果子问题之间不独立，则分治法需要重复地解决公共的子问题，造成效率低下的结果。

根据前面我们对分治法的分析，你一定能迅速联想到递归。分治法需要递归地分解问题，再去解决问题。因此，**分治法在每轮递归上，都包含了分解问题、解决问题和合并结果这 3 个步骤。**

为了让大家对分治法有更清晰地了解，我们以二分查找为例，看一下分治法如何使用。关于分治法在排序中的使用，我们会在第 11 课时中讲到。查找问题指的是，在一个有序的数列中，判断某个待查找的数字是否出现过。二分查找，则是利用分治法去解决查找问题。通常二分查找需要一个前提，那就是输入的数列是有序的。

二分查找的思路比较简单，步骤如下：

1. 选择一个标志 i 将集合 L 分为二个子集合，一般可以使用中位数；
2. 判断标志 L(i) 是否能与要查找的值 des 相等，相等则直接返回结果；
3. 如果不相等，需要判断 L(i) 与 des 的大小；
4. 基于判断的结果决定下步是向左查找还是向右查找。如果向某个方向查找的空间为 0，则返回结果未查到；

5. 回到步骤 1。

我们对二分查找的复杂度进行分析。二分查找的最差情况是，不断查找到最后 1 个数字才完成判断。那么此时需要的最大的复杂度就是 O(logn)。

分治法的案例

下面我们一起来看一个例子。**在数组 { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 } 中，查找 8 是否出现过。**

首先判断 8 和中位数 5 的大小关系。因为 8 更大，所以在更小的范围 6, 7, 8, 9, 10 中继续查找。此时更小的范围的中位数是 8。由于 8 等于中位数 8，所以查找到并打印查找到的 8 对应在数组中的 index 值。如下图所示。

开始

从代码实现的角度来看，我们可以采用两个索引 low 和 high，确定查找范围。最初 low 为 0，high 为数组长度减 1。在一个循环体内，判断 low 到 high 的中位数与目标变量 targetNumb 的大小关系。根据结果确定向左走 (high = middle - 1) 或者向右走 (low = middle + 1) ，来调整 low 和 high 的值。直到 low 反而比 high 更大时，说明查找不到并跳出循环。我们给出代码如下：

```
public static void main(String[] args) {
        // 需要查找的数字
        int targetNumb = 8;
```

```
        // 目标有序数组
        int[] arr = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int middle = 0;
        int low = 0;
        int high = arr.length - 1;
    int isfind = 0;

        while (low <= high) {
                middle = (high + low) / 2;
                if (arr[middle] == targetNumb) {
                        System.out.println(targetNumb + " 在数组中,下标值为: " + middle);
            isfind = 1;
                        break;
                } else if (arr[middle] > targetNumb) {
                        // 说明该数在low~middle之间
                        high = middle - 1;
                } else {
                        // 说明该数在middle~high之间
                        low = middle + 1;
                }
    }
    if (isfind == 0) {
                        System.out.println("数组不含 " + targetNumb);
        }
}
```

我们基于这个例子，可以对它进行一些经验和规律的总结，这些经验会辅助大家在面试时找到解题思路。

1. **二分查找的时间复杂度是 O(logn)，这也是分治法普遍具备的特性。当你面对某个代码题，而且约束了时间复杂度是 O(logn) 或者是 O(nlogn) 时，可以想一下分治法是否可行。**

2. **二分查找的循环次数并不确定。一般是达到某个条件就跳出循环。因此，编码的时候，多数会采用 `while` 循环加 `break` 跳出的代码结构。**
3. **二分查找处理的原问题必须是有序的。因此，当你在一个有序数据环境中处理问题时，可以考虑分治法。相反，如果原问题中的数据并不是有序的，则使用分治法的可能性就会很低了。**

以上 3 点经验和规律的总结，可以帮助你快速找到解决方案，做好技术选型。在实际工作和参加面试时，都是非常重要的经验。

练习题

最后，我们给出一个进阶的问题，供大家练习。题目如下：

在一个有序数组中，查找出第一个大于 9 的数字，假设一定存在。例如，`arr = { -1, 3, 3, 7, 10, 14, 14 };` 则返回 10。

在这里提醒一下，带查找的目标数字具备这样的性质：

第一，它比 9 大；

第二，它前面的数字（除非它是第一个数字），比 9 小。

因此，当我们作出向左走或向右走的决策时，必须满足这两个条件。

```
public static void main(String[] args) {
        int targetNumb = 9;
        // 目标有序数组
        int[] arr = { -1, 3, 3, 7, 10, 14, 14 };
        int middle = 0;
        int low = 0;
        int high = arr.length - 1;
        while (low <= high) {
```

```
                middle = (high + low) / 2;
                if (arr[middle] > targetNumb && (middle == 0 || arr[middle - 1] <= targetNumb)) {
                        System.out.println("第一个比 " + targetNumb + " 大的数字是 " + arr[middle]);
                        break;
                } else if (arr[middle] > targetNumb) {
                        // 说明该数在low~middle之间
                        high = middle - 1;
                } else {
                        // 说明该数在middle~high之间
                        low = middle + 1;
                }
        }
}
```

总结

分治法经常会用在海量数据处理中。这也是它显著区别于遍历查找方法的优势。**在面对陌生问题时，需要注意原问题的数据是否有序，预期的时间复杂度是否带有 `logn` 项，是否可以通过小问题的答案合并出原问题的答案。如果这些先决条件都满足，你就应该第一时间想到分治法。**

上一页 下一页