

38|分治算法：谈一谈大规模计算框架MapReduce中的分治思想

MapReduce是Google大数据处理的三驾马车之一，另外两个是GFS和Bigtable。它在倒排索引、PageRank计算、网页分析等搜索引擎相关的技术中都有大量的应用。

尽管开发一个MapReduce看起来很高深，感觉跟我们遥不可及。实际上，万变不离其宗，它的本质就是我们今天要学的这种算法思想，分治算法。

如何理解分治算法？

为什么说MapReduce的本质就是分治算法呢？我们先来看，什么是分治算法？

分治算法（divide and conquer）的核心思想其实就是四个字，分而治之，也就是将原问题划分成 n 个规模较小，并且结构与原问题相似的子问题，递归地解决这些子问题，然后再合并其结果，就得到原问题的解。

这个定义看起来有点类似递归的定义。关于分治和递归的区别，我们在排序（下）的时候讲过，分治算法是一种处理问题的思想，递归是一种编程技巧。实际上，分治算法一般都比较适合用递归来实现。分治算法的递归实现中，每一层递归都会涉及这样三个操作：

- 分解：将原问题分解成一系列子问题；
- 解决：递归地求解各个子问题，若子问题足够小，则直接求解；
- 合并：将子问题的结果合并成原问题。

分治算法能解决的问题，一般需要满足下面这几个条件：

- 原问题与分解成的小问题具有相同的模式；
- 原问题分解成的子问题可以独立求解，子问题之间没有相关性，这一点是分治算法跟动态规划的明显区别，等我们讲到动态规划的时候，会详细对比这两种算法；
- 具有分解终止条件，也就是说，当问题足够小时，可以直接求解；
- 可以将子问题合并成原问题，而这个合并操作的复杂度不能太高，否则就起不到减小算法总体复杂度的效果了。

分治算法应用举例分析

理解分治算法的原理并不难，但是要想灵活应用并不容易。所以，接下来，我会带你用分治算法解决我们在讲排序的时候涉及的一个问题，加深你对分治算法的理解。

还记得我们在排序算法里讲的数据的有序度、逆序度的概念吗？我当时讲到，我们用有序度来表示一组数据的有序程度，用逆序度表示一组数据的无序程度。

假设我们有 n 个数据，我们期望数据从小到大排列，那完全有序的数据的有序度就是 $n(n-1)/2$ ，逆序度等于0；相反，倒序排列的数据的有序度就是0，逆序度是 $n(n-1)/2$ 。除了这两种极端情况外，我们通过计算有序对或者逆序对的个数，来表示数据的有序度或逆序度。

2, 4, 3, 1, 5, 6 逆序对个数: 4

(2, 1) (4, 3) (4, 1) (3, 1)

我现在的的问题是，如何编程求出一组数据的有序对个数或者逆序对个数呢？因为有序对个数和逆序对个数的求解方式是类似的，所以你可以只思考逆序对个数的求解方法。

最笨的方法是，拿每个数字跟它后面的数字比较，看有几个比它小的。我们把比它小的数字个数记作 k ，通过这样的方式，把每个数字都考察一遍之后，然后对每个数字对应的 k 值求和，最后得到的总和就是逆序对个数。不过，这样操作的时间复杂度是 $O(n^2)$ 。那有没有更加高效的处理方法呢？

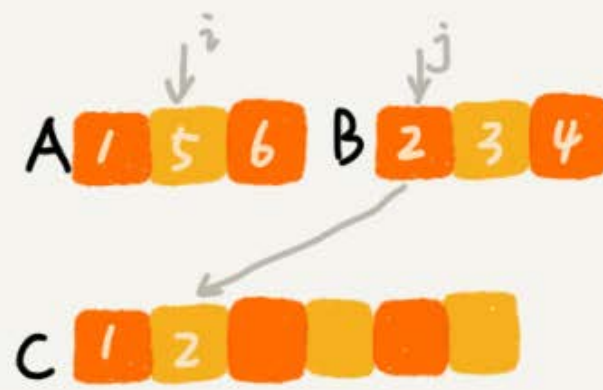
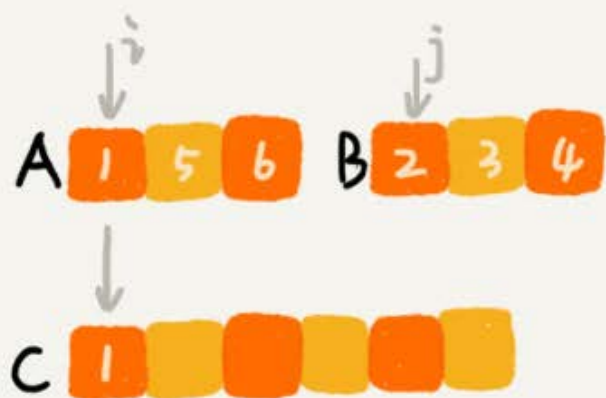
我们用分治算法来试试。我们套用分治的思想来求数组 A 的逆序对个数。我们可以将数组分成前后两半 $A1$ 和 $A2$ ，分别计算 $A1$ 和 $A2$ 的逆序对个数 $K1$ 和 $K2$ ，然后再计算 $A1$ 与 $A2$ 之间的逆序对个数 $K3$ 。那数组 A 的逆序对个数就等于 $K1+K2+K3$ 。

我们前面讲过，使用分治算法其中一个要求是，子问题合并的代价不能太大，否则就起不了降低时间复杂度的效果了。那回到这个问题，如何快速计算出两个子问题 $A1$ 与 $A2$ 之间的逆序对个数呢？

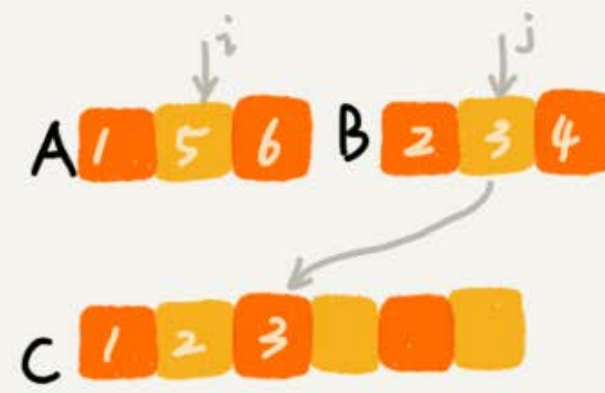
这里就要借助归并排序算法了。你可以先试着想想，如何借助归并排序算法来解决呢？

归并排序中有一个非常关键的操作，就是将两个有序的小数组，合并成一个有序的数组。实际上，在这个合并的过程中，我们就可以计算这两个小数组的逆序对个数了。每次合并操作，我们都计算逆序对个数，把这些计算出来的逆序对个数求和，就是这个数组的逆序对个数了。

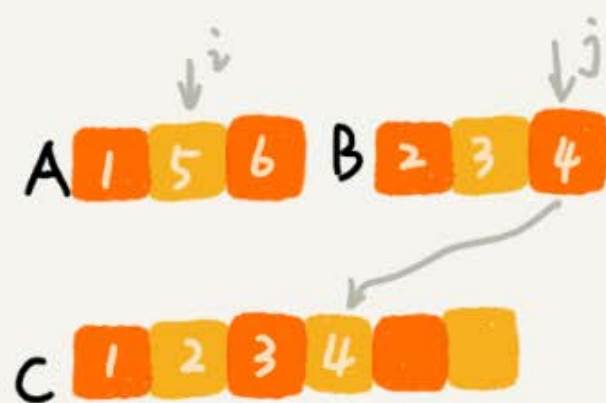
统计A和B之间的逆序对个数 $0+3+3=6$



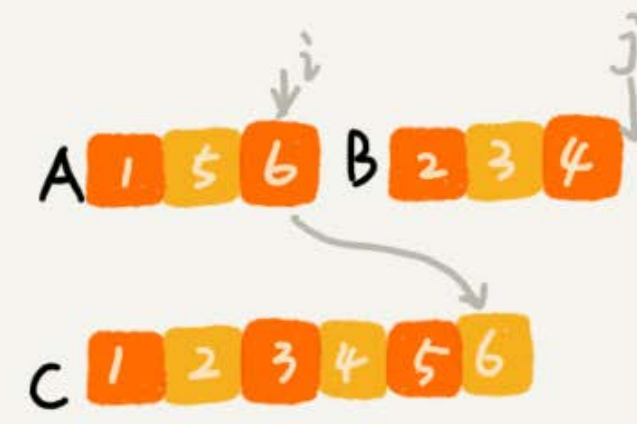
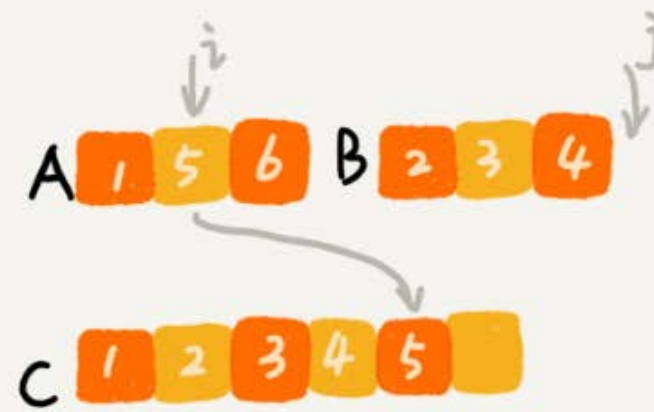
A中比2大的有2个



A中比3大的有2个



A中比4大的有2个



尽管我画了张图来解释，但是我个人觉得，对于工程师来说，看代码肯定更好理解一些，所以我们把这个过程翻译成了代码，你可以结合着图和文字描述一起看下。

```
private int num = 0; // 全局变量或者成员变量

public int count(int[] a, int n) {
    num = 0;
    mergeSortCounting(a, 0, n-1);
    return num;
}

private void mergeSortCounting(int[] a, int p, int r) {
    if (p >= r) return;
    int q = (p+r)/2;
    mergeSortCounting(a, p, q);
    mergeSortCounting(a, q+1, r);
    merge(a, p, q, r);
}

private void merge(int[] a, int p, int q, int r) {
    int i = p, j = q+1, k = 0;
    int[] tmp = new int[r-p+1];
    while (i <= q && j <= r) {
        if (a[i] <= a[j]) {
            tmp[k++] = a[i++];
        } else {
            num += (q-i+1); // 统计p-q之间，比a[j]大的元素个数
            tmp[k++] = a[j++];
        }
    }
    while (i <= q) { // 处理剩下的
        tmp[k++] = a[i++];
    }
    while (j <= r) { // 处理剩下的
        tmp[k++] = a[j++];
    }
    for (i = 0; i <= r-p; ++i) { // 从tmp拷贝回a
        a[p+i] = tmp[i];
    }
}
```

有很多同学经常说，某某算法思想如此巧妙，我是怎么也想不到的。实际上，确实是的。有些算法确实非常巧妙，并不是每个人短时间都能想到的。比如这个问题，并不是每个人都能想到可以借助归并排序算法来解决，不夸张地说，如果之前没接触过，绝大部分人都想不到。但是，如果我告诉你可以借助归并排序算法来解决，那你就应该要想到如何改造归并排序，来求解这个问题了，只要你能做到这一点，我觉得就很棒了。

关于分治算法，我这还有两道比较经典的问题，你可以自己练习一下。

- 二维平面上有n个点，如何快速计算出两个距离最近的点对？
- 有两个n*n的矩阵A，B，如何快速求解两个矩阵的乘积C=A*B？

分治思想在海量数据处理中的应用

分治算法思想的应用是非常广泛的，并不仅限于指导编程和算法设计。它还经常用在海量数据处理的场景中。我们前面讲的数据结构和算法，大部分都是基于内存存储和单机处理。但是，如果要处理的数据量非常大，没法一次性放到内存中，这个时候，这些数据结构和算法就无法工作了。

比如，给**10GB**的订单文件按照金额排序这样一个需求，看似是一个简单的排序问题，但是因为数据量大，有**10GB**，而我们的机器的内存可能只有**2、3GB**这样子，无法一次性加载到内存，也就无法通过单纯地使用快排、归并等基础算法来解决了。

要解决这种数据量大到内存装不下的问题，我们就可以利用分治的思想。我们可以将海量的数据集合根据某种方法，划分为几个小的数据集合，每个小的数据集合单独加载到内存来解决，然后再将小数据集合合并成大数据集合。实际上，利用这种分治的处理思路，不仅仅能克服内存的限制，还能利用多线程或者多机处理，加快处理的速度。

比如刚刚举的那个例子，给**10GB**的订单排序，我们就可以先扫描一遍订单，根据订单的金额，将**10GB**的文件划分为几个金额区间。比如订单金额为**1**到**100**元的放到一个小文件，**101**到**200**之间的放到另一个文件，以此类推。这样每个小文件都可以单独加载到内存排序，最后将这些有序的小文件合并，就是最终有序的**10GB**订单数据了。

如果订单数据存储存储在类似**GFS**这样的分布式系统上，当**10GB**的订单被划分成多个小文件的时候，每个文件可以并行加载到多台机器上处理，最后再将结果合并在一起，这样并行处理的速度也加快了很多。不过，这里有一个点要注意，就是数据的存储与计算所在的机器是同一个或者在网络中靠的很近（比如一个局域网内，数据存取速度很快），否则就会因为数据访问的速度，导致整个处理过程不但不会变快，反而有可能变慢。

你可能还有印象，这个就是我在讲线性排序的时候举的例子。实际上，在前面已经学习的课程中，我还讲了很多利用分治思想来解决的问题。

解答开篇

分治算法到此就讲完了，我们现在来看下开篇的问题，为什么说**MapReduce**的本质就是分治思想？

我们刚刚举的订单的例子，数据有**10GB**大小，可能给你的感受还不强烈。那如果我们要处理的数据是**1T、10T、100T**这样子的，那一台机器处理的效率肯定是非常低的。而对于谷歌搜索引擎来说，网页爬取、清洗、分析、分词、计算权重、倒排索引等等各个环节中，都会面对如此海量的数据（比如网页）。所以，利用集群并行处理显然是大势所趋。

一台机器过于低效，那我们就把任务拆分到多台机器上来处理。如果拆分之后的小任务之间互不干扰，独立计算，最后再将结果合并。这不就是分治思想吗？

实际上，**MapReduce**框架只是一个任务调度器，底层依赖**GFS**来存储数据，依赖**Borg**管理机器。它从**GFS**中拿数据，交给**Borg**中的机器执行，并且时刻监控机器执行的进度，一旦出现机器宕机、进度卡壳等，就重新从**Borg**中调度一台机器执行。

尽管**MapReduce**的模型非常简单，但是在**Google**内部应用非常广泛。它除了可以用来处理这种数据与数据之间存在关系的任务，比如**MapReduce**的经典例子，统计文件中单词出现的频率。除此之外，它还可以用来处理数据与数据之间没有关系的任务，比如对网页分析、分词等，每个网页可以独立的分析、分词，而这两个网页之间并没有关系。网页几十亿、上百亿，如果单机处理，效率低下，我们就可以利用**MapReduce**提供的高可靠、高性能、高容错的并行计算框架，并行地处理这几十亿、上百亿的网页。

内容小结

今天我们讲了一种应用非常广泛的算法思想，分治算法。

分治算法用四个字概括就是“分而治之”，将原问题划分成**n**个规模较小而结构与原问题相似的子问题，递归地解决这些子问题，然后再合并其结果，就得到原问题的解。这个思想非常简单、好理解。

今天我们讲了两组分治算法的典型的应用场景，一个是用来指导编码，降低问题求解的时间复杂度，另一个是解决海量数据处理问题。比如**MapReduce**本质上就是利用了分治思想。

我们也时常感叹Google的创新能力如此之强，总是在引领技术的发展。实际上，创新并非离我们很远，创新的源泉来自对事物本质的认识。无数优秀架构设计的思想来源都是基础的数据结构和算法，这本身就是算法的一个魅力所在。

课后思考

我们前面讲过的数据结构、算法、解决思路，以及举的例子中，有哪些采用了分治算法的思想呢？除此之外，生活、工作中，还有没有其他用到分治算法的地方呢？你可以自己回忆、总结一下，这对你将零散的知识提炼成体系非常有帮助。

欢迎留言和我分享，也欢迎点击“请朋友读”，把今天的内容分享给你的好友，和他一起讨论、学习。



数据结构与算法之美

为工程师量身打造的数据结构与算法私教课

王争

前 Google 工程师



新版升级：点击「👤请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

精选留言：

- 三木子 2018-12-19 22:27:02

在统计方面比较多，比如统计我国人口，要知道我国人口就要先知道每个省人口，要知道省人口就要知道每个市人口，要知道市人口就要知道每个区县人口，直到村社区，然后汇总求的总人数。[6赞]

- 刘远通 2018-12-20 10:03:38

第一个求最近的点对
分成两块 单独求其中一块点对最小距离
然后求这两块之间点对的最小距离 通过一些排序和删除 可以减少到6个点之间比较 很神奇

第二个矩阵计算
v.斯特拉森提出了2*2分块矩阵的计算公式 从原来的8次乘法 缩减到了7次
当n规模很大的时候 缩减效果就很明显 $(7/8)^{\log n}$ [5赞]

- Yves 2018-12-19 12:10:10

代码略有问题：1, num += (q - i + 1), 应该是在 a[i] <= a[j] 这个条件分支里面； 2, while (i <= q) 里面不应该有 num += (q - i + 1), 3, 最后的修改原数组迭代条件应该是 i < r - p + 1 而不是 i < r - p 。

```
private void merge(int[] a, int p, int q, int r) {  
    int i = p, j = q + 1, k = 0;  
    int[] tmp = new int[r - p + 1];  
    while (i <= q && j <= r) {  
        if (a[i] <= a[j]) {  
            tmp[k++] = a[i++];  
        } else {  
            num += (q - i + 1);  
            tmp[k++] = a[j++];  
        }  
    }  
    while (i <= q) {  
        tmp[k++] = a[i++];  
    }  
    while (j <= r) {  
        tmp[k++] = a[j++];  
    }  
    for (i = 0; i < r - p + 1; ++i) {
```



```
    a[p + i] = tmp[i];
}
} [4赞]
```

作者回复2018-12-19 19:40:48

是的 大家代码直接看这位同学的 我晚点改正下。写的时候匆忙 不好意思

- Williamzhang 2018-12-20 10:22:11

第一个留言有问题可以再理解一下，不要误导后边人，作者的num+=语句位置正确 [3赞]

作者回复2018-12-21 10:07:54

哈哈 终于有人看懂了 我的跟留言那位同学的思路不一样而已 他的更简洁些

- Sharry 2018-12-20 11:38:35

使用归并排序求逆序度，实在是太妙了！老师的代码一点儿问题都没有，(j-q-1)记录的是比子序列 A 中当前即将添加到 tmp 数组中的元素的逆序对的个数，看到留言里小伙伴说应该放在下面的 else 里，可能需要再斟酌斟酌 [2赞]

- 不上进的码农 2018-12-19 08:19:56

```
if (a[i] <= a[j]) {
    num += (j - q - 1);
    tmp[k++] = a[i++];
} else {
    tmp[k++] = a[j++];
}
```

我想了想，这段代码应该求的是有序对而不是逆序对吧 [2赞]

作者回复2018-12-19 09:38:37

逆序对

- 渊凝 2019-01-11 11:39:48

老师，我有两个疑问：

给 10GB 的订单排序，我们就可以先扫描一遍订单

- 1.场景中描述的机器内存只有2、3GB，我理解的是直接加载文件内存应该不够用来扫描一次10GB订单文件，对吗？如果不能，那应该怎么扫描呢？
- 2.如果用buffer来缓存扫描结果的话，即使能扫描完成，又该怎么对文件根据金额区间进行分割呢？ [1赞]

- MIAN-勉 2018-12-27 21:36:43

把归并排序merge方法的参数列表 由merge(int[] a, int p, int q, int r) 改为 merge(int[] a, int low, int middle, int high) 更容易理解，小细节，哈哈 [1赞]

• ALAN 2018-12-19 20:22:20

老师，你好，有一个建议，就是对于代码每一行里不是很显然易懂的地方，能否注释下此行代码的作用，不然有时看了代码也不知为啥这样写。 [1赞]

• Smallfly 2018-12-19 08:47:19

「创新并非离我们很远，创新的源泉来自对事物本质的认识。无数优秀架构设计的思想来源都是基础的数据结构和算法，这本身就是算法的一个魅力所在。」

这句话讲的太好啦。各种前端框架层出不穷，本质的东西，也是基本都没有变。

与其最新，不如求本。 [1赞]

• Uncle IrNing 2019-01-26 02:41:20

老师啊，能不能把变量名取得有意义一点儿？比如p改成left，r改成right，q改成mid，这样更方便阅读代码啊

• 睡痴儿 2019-01-23 09:29:13

树的基本思想其实就是分治。我觉得时间复杂度为logn的，或多或少都会与分治有关。

• luo 2019-01-17 10:28:10

老师 你说的转算法，岁数不大愿意学习的岁数不大应该是小于多少岁为好

作者回复2019-01-25 16:46:32

这个看个人心态吧，多大都不晚。不过我个人觉得工作5年内吧。

• qinggeouye 2019-01-15 01:19:07

第 12 讲 归并排序和快速排序都用到了分治思想，以及思考题
<https://time.geekbang.org/column/article/41913>

• cw 2019-01-13 21:13:19

冲鸭

• Geek_fbe6fe 2019-01-08 11:29:55

还是写代码理解的快
package baseMind

```
import (  
    "fmt"
```

)

var totalNum int

```
func CountReversed() {
list := []int{1, 2, 43, 344, 3, 46, 75}
//list := []int{5, 2, 43, 344, 3, 46, 75}
mergeSortCount(list, 0, len(list)-1)
fmt.Println(totalNum)
}
func mergeSortCount(a []int, start int, end int) {
//return
if start >= end {
return
}
low := start
min := (start + end) / 2
mergeSortCount(a, low, min)
mergeSortCount(a, min+1, end)
mergeCount(a, low, min, end)
}
func mergeCount(a []int, low int, min int, end int) {
tmp := make([]int, end-low+1)
i := low
k := 0
j := min + 1
//获取左半边和右半边的逆序度
for i <= min && j <= end {
if a[i] < a[j] {
tmp[k] = a[i]
k++
i++
```

```
    } else {
        totalNum += min - i + 1
        tmp[k] = a[j]
        k++
        j++
    }
}

//处理左边剩下
//fmt.Println(i, min)
for i <= min {
    tmp[k] = a[i]
    k++
    i++
}

//处理右边剩下
//fmt.Println(j, end)
for j <= end {
    tmp[k] = a[j]
    k++
    j++
}

fmt.Println(tmp, low, min, end)

//将数据拷贝回a
for n := 0; n < end-low+1; n++ {
    //fmt.Println(n)
    a[low+n] = tmp[n]
}

fmt.Println(a)

fmt.Println("num:", totalNum)
}
```

其实，分析递归应该从最小最底层的递归进行分析，要不然很容易产生误解，归并的最底层只有两个元素所以可以用这种方式做逆序计算，合并的时候两个小组已经是有序的所以也可以用合并的方式做合并，真的挺神奇的，换一个方式角度思考问题就解决了。

38|分治算法：谈一谈大规模计算框架MapReduce中的分治思想

- spark 2019-01-06 11:42:07

归并排序忘记了，又跑头前面看一下，不然看不懂了……忘的好快……

- Geek_fbe6fe 2019-01-03 10:42:08

老师，使用归并排序算有序度，分成两个小数组必须是有序的，是吗？好疑惑这样子应用场景不是很小？

作者回复2019-01-04 09:52:45

不 你有点误解了 你最好还是回过头去再看下归并排序那一节课

- 苏雅拉 2019-01-02 08:16:35

微积分是分治思想鼻祖

- 唯她命 2018-12-28 23:06:23

老师 一开始逆序度不是4吗，最后怎么成了6了

作者回复2019-01-02 16:36:59

不是同一个例子 你再仔细看看