第 10 章　海量数据处理

计算机硬件的扩容确实可以极大地提高程序的处理速度，但考虑到技术、成本等方面的因素，它并非只有一条途径。而随着互联网技术的发展，机器学习、深度学习、大数据、人工智能、云计算、物联网和移动通信技术的发展，每时每刻，数以亿万计的用户产生着数量巨大的信息，海量数据时代已经来临。因为通过对海量数据的挖掘能有效地揭示用户的行为模式，加深对用户需求的理解，提取用户的集体智慧，从而为研发人员决策提供依据，提升产品用户体验，进而占领市场。所以当前各大互联网公司的研究工作都将重点放在了海量数据分析上。但是，只寄希望于硬件扩容是很难满足海量数据的分析需求，如何利用现有条件进行海量信息处理已经成为各大互联网公司亟待解决的问题。所以，海量信息处理正日益成为当前程序员笔试、面试中一个新的亮点。

不同于常规量级数据中提取信息，在海量数据中提取信息，会存在以下几个方面的问题。首先，数据量过大，数据中什么情况都可能存在，如果信息数量只有 20 条，那么人工就可以逐条进行查找、比对。可是当数据规模扩展到上百条、数千条、数亿条甚至更多时，只通过人工已经无法解决存在的问题，必须通过工具或者程序进行处理。其次，对海量数据信息处理，还需要有良好的软硬件配置，合理使用工具，合理分配系统资源。通常情况下，如果需要处理的数据量非常大，超过了 TB 级，小型机、大型工作站是要考虑的，普通的计算机如果有好的处理方法也可以考虑，如通过联机做成工作集群。最后，对海量数据信息进行处理时，要求很好的处理方法和技巧，如何进行数据挖掘算法的设计以及如何进行数据的存储访问等都是研究的难点。

针对海量数据的处理，可以使用的方法非常多，常见的方法有 Hash 法、Bit-map（位图）法、Bloom filter 法、数据库优化法、倒排索引法、外排序法、Trie 树、堆、双层桶法以及 MapReduce 法等。其中，Hash 法、Bit-map（位图）法、Trie 树和堆等方法的考查频率最高、使用范围最为广泛，是读者需要重点掌握的方法。

10.1 如何从大量的 url 中找出相同的 url

难度系数：★★★★☆　　**被考查系数：**★★★★☆

题目描述：

给定 a、b 两个文件，各存放 50 亿个 url，每个 url 各占 64 个字节，内存限制是 4GB，请找出 a、b 两个文件共同的 url。

分析与解答：

因为每个 url 需要占 64 个字节，所以 50 亿个 url 占用空间的大小为 50 亿×64=5GB×64=320GB。由于内存大小只有 4GB，因此不可能一次性把所有的 url 都加载到内存中处理。对于这个类型的题目，一般都需要使用分治法，即把一个文件中的 url 按照某一特征分成多个文件，使得每个文件的内容都小于 4GB，这样就可以把这个文件一次性读到内存中进行处理了。对

于本题而言，实现思路为以下几点。

1）遍历文件 a，对遍历到的 url 求 hash(url)%500，根据计算结果把遍历到的 url 分别存储到 a0、a1、a2、…、a499（将计算结果为 i 的 url 存储到文件 ai 中）中，这样每个文件的大小约为 600MB。当某一个文件中的 url 大小超过 2GB 时，可以按照类似的思路把这个文件继续分为更小的子文件（例如，如果 a1 大小超过 2GB，那么可以把文件继续分成 a11、a12 等）。

2）使用同样的方法遍历文件 b，把文件 b 中的 url 分别存储到文件 b0、b1、…、b499 中。

3）通过上面的划分，与 ai 中 url 相同的 url 一定在 bi 中。由于 ai 与 bi 中所有的 url 大小都不会超过 4GB，因此可以把它们同时读入内存中进行处理。具体思路：遍历文件 ai，把遍历到的 url 存入到 hash_set 中，接着遍历文件 bi 中的 url，如果这个 url 在 hash_set 中存在，那么说明这个 url 是这两个文件共同的 url，可以把这个 url 保存到另外一个单独的文件中。当把文件 a0～a499 都遍历完成后，就找到了两个文件共同的 url。

10.2 如何从大量数据中找出高频词

难度系数：★★★★☆ **被考查系数：★★★★★**

题目描述：

有一个 1GB 大小的文件，文件里面每一行是一个词，每个词的大小不超过 16 个字节，内存大小限制是 1MB，要求返回频数最高的 100 个词。

分析与解答：

由于文件大小为 1GB，而内存大小只有 1MB，因此不可能一次把所有的词读到内存中处理，需要采用分治的方法，把一个大的文件分解成多个小的子文件，从而保证每个文件的大小都小于 1MB，进而可以直接被读取到内存中处理。具体的思路如下：

1）遍历文件，对遍历到的每一个词，执行如下 hash 操作：hash(x)%2000，将结果为 i 的词存放到文件 ai 中。通过这个分解步骤，可以使每个子文件的大小为 400KB 左右。如果这个操作后某个文件的大小超过 1MB 了，那么就可以采用相同的方法对这个文件继续分解，直到文件小于 1MB 为止。

2）统计出每个文件中出现频率最高的 100 个词。最简单的方法是使用 hash_map 来实现，先遍历文件中的所有词，然后对于遍历到的词，如果在 hash_map 中不存在，那么把这个词存入 hash_map 中（键为这个词，值为 1）；如果这个词在 hash_map 中已经存在了，那么把这个词对应的值加 1。遍历完后可以非常容易地找出出现频率最高的 100 个词。

3）第 2）步找出了每个文件出现频率最高的 100 个词，这一步可以通过维护一个小顶堆来找出所有词中出现频率最高的 100 个。具体方法：先遍历第一个文件，把第一个文件中出现频率最高的 100 个词构建成一个小顶堆如果第一个文件中词的个数小于 100，则可以继续遍历第二个文件，直到构建好包含 100 个结点的小顶堆为止。然后继续遍历，如果遍历到的词的出现次数大于堆顶上词的出现次数，那么可以用新遍历到的词替换堆顶的词，再重新调整这个堆为小顶堆。当遍历完所有文件后，这个小顶堆中的词就是出现频率最高的 100 个词。当然这一步也可以采用类似归并排序的方法把所有文件中出现频率最高的 100 个词排序，最终找出出现频率最高的 100 个词。

引申：怎么在海量数据中找出重复次数最多的一个。

分析与解答：前面的算法是求解 top100，而这道题目只是求解 top1，可以使用同样的思路来求解。唯一不同的是，在求解出每个文件中出现次数最多的数据后，接下来不需要通过小顶堆来找出出现次数最多的数，只需要使用一个变量就可以完成。方法很简单，此处不再赘述。

10.3 如何找出某一天访问百度网站最多的 IP

难度系数：★★★★☆　　**被考查系数：**★★★★★

题目描述：

现有海量日志数据保存在一个超级大的文件中，该文件无法直接读入内存，要求从中提取某天访问百度次数最多的那个 IP。

分析与解答：

由于这道题只关心某一天访问百度最多的 IP，因此可以首先对文件进行一次遍历，把这一天访问百度的 IP 的相关信息记录到一个单独的文件中。接下来可以用上一题找出高频词介绍的方法来求解。由于求解思路是一样的，这里就不再详细介绍了。唯一需要确定的是，把一个大文件分为几个小文件比较合适。以 IPv4 为例，由于一个 IP 地址占用 32 位，因此最多会有 2^{32} 种取值情况。如果使用 hash(IP)%1024，那么就把海量 IP 日志分别存储到 1024 个小文件中。这样的话，每个小文件最多包含 4MB 个 IP 地址；如果使用 2048 个小文件，那么每个文件会最多包含 2MB 个 IP 地址。因此，对于这类题目而言，首先需要确定可用内存的大小，然后确定数据的大小。由这两个参数就可以确定 hash 函数应该怎么设置才能保证每个文件的大小都不超过内存的大小，从而可以保证每个小文件都能被一次性加载到内存中。

10.4 如何在大量的数据中找出不重复的整数

难度系数：★★★★☆　　**被考查系数：**★★★★★

题目描述：

在 2.5 亿个整数中找出不重复的整数。注意：内存不足以容纳这 2.5 亿个整数。

分析与解答：

由于这道题目与前面的题目类似，也是无法一次性把所有数据加载到内存中，因此也可以采用类似的方法求解。

方法一：分治法

采用 Hash 函数的方法，把这 2.5 亿个数划分到更小的文件中，从而保证每个文件的大小不超过可用内存的大小。然后对每个小文件而言，所有的数据都可以一次性的被加载到内存中，因此可以使用 hash_map 或 hash_set 来找到每个小文件中不重复的整数。当处理完所有的文件后就可以找出这 2.5 亿个整数中所有不重复的整数。

方法二：位图法

对于整数相关的算法的求解，位图法是一种非常实用的算法。对本题而言，如果可用的内存空间超过 1GB 就可以使用这种方法。具体思路：假设整数占用 4 个字节（如果占用 8 个字节，则求解思路类似，只不过需要占用更大的内存），4 个字节也就是 32 位，可以表示的整

数个数为 2^{32}。由于本题只查找不重复的数，而不关心具体数字出现的次数，因此可以分别使用 2 个位（bit）来表示各个数字的状态：用 00 表示这个数字没有出现过，01 表示出现过 1 次，10 表示出现了多次，11 暂不使用。

根据上面的逻辑，在遍历这 2.5 亿个整数时，如果这个整数对应位图中的位是 00，那么就修改成 01；如果是 01，则修改为 10；如果是 10，则保持原值不变。这样当所有数据遍历完成后，可以再遍历一次位图，位图中为 01 的对应数字就是没有重复的数字。

10.5 如何在大量的数据中判断一个数是否存在

难度系数：★★★★☆ **被考查系数：★★★★☆**

题目描述：

在 2.5 亿个整数中判断一个数是否存在。注意：内存不足以容纳这 2.5 亿个整数。

分析与解答：

显然 2.5 亿数据量太大，不可能一次性把所有的数据都加载到内存中，那么最容易想到的方法就是分治法。

方法一：分治法

对于大数据相关的算法题，分治法是一个非常好的方法。针对这道题而言，主要的思路：首先根据实际可用内存的情况，确定一个 hash()函数，比如 hash(value)%1000，通过这个 hash()函数可以把这 2.5 亿个数字划分到 1000 个文件中（a1，a2，…，a1000）；然后再对待查找的数字使用相同的 hash()函数求出 hash 值，假设计算出的 hash 值为 i，如果这个数存在，那么它一定在文件 ai 中。通过这种方法就可以把题目的问题转换为文件 ai 中是否存在这个数。那么在接下来的求解过程中可以选用的思路比较多，如下所列：

1）由于划分后的文件比较小，所以可以直接被装载到内存中，先把文件中所有的数字都保存到 hash_set 中，然后判断待查找的数字是否存在。

2）如果这个文件中的数字占用的空间还是太大，那么就可以用相同的方法把这个文件继续划分为更小的文件，然后确定待查找的数字可能存在的文件，最后在相应的文件中继续查找。

方法二：位图法

对于这类判断数字是否存在、判断数字是否重复的问题，位图法是一种非常高效的方法。这里以 32 位整型为例，它可以表示数字的个数为 2^{32}。可以申请一个位图，让每个整数对应位图中的一个位，这样 2^{32} 个数需要位图的大小为 512MB。具体实现的思路：申请一个 512MB 大小的位图，并把所有的位都初始化为 0；接着遍历所有的整数，对遍历到的数字，把相应位置上的位设置为 1。最后判断待查找的数对应的位图上的值是多少，如果是 0，则表示这个数字不存在，如果是 1，则表示这个数字存在。

10.6 如何查询最热门的查询串

难度系数：★★★★☆ **被考查系数：★★★★★**

题目描述：

搜索引擎会通过日志文件把用户每次检索使用的所有查询串都记录下来，每个查询串的

长度为 1～255 字节。

假设目前有 1000 万个记录（这些查询串的重复度比较高，虽然总数是 1000 万，但如果除去重复后，则不超过 300 万个），请统计最热门的 10 个查询串（一个查询串的重复度越高，说明查询它的用户越多，也就是越热门），要求使用的内存不能超过 1GB。

分析与解答：

从题目中可以发现，每个查询串最长为 255 个字节，1000 万个字符串需要占用 2.55GB 内存，因此无法把所有的字符串全部读入到内存中处理。对于这种类型的题目，分治法是一个非常实用的方法。

方法一：分治法

对字符串设置一个 hash()函数，通过这个 hash()函数把字符串划分到更多更小的文件中，从而保证每个小文件中的字符串都可以直接被加载到内存中处理，然后求出每个文件中出现次数最多的 10 个字符串；最后通过一个小顶堆统计出所有文件中出现最多的 10 个字符串。

从功能角度出发，这种方法是可行的，但是因为需要对文件遍历两次，而且 hash()函数也需要被调用 1000 万次，所以性能不是很好。针对这道题的特殊性，下面介绍另外一种性能较好的方法。

方法二：hash_map 法

虽然字符串的总数比较多，但是字符串的种类不超过 300 万个，因此可以考虑把所有字符串出现的次数保存在一个 hash_map 中（键为字符串，值为字符串出现的次数）。hash_map 所需要的空间为 300 万×(255+4)=3M×259=777MB（其中，4 表示用来记录字符串出现次数的整数所占用的 4 个字节）。由此可见，1GB 的内存空间是足够用的。基于以上的分析，本题的求解思路如下：

1）遍历字符串，如果字符串在 hash_map 中不存在，则直接存入 hash_map 中，键为这个字符串，值为 1。如果字符串在 hash_map 中已经存在，则把对应的值直接加 1。这一步操作的时间复杂度为 O(N)，其中 N 为字符串的数量。

2）在第一步的基础上找出出现频率最高的 10 个字符串。可以通过小顶堆的方法来完成，遍历 hash_map 的前 10 个元素，并根据字符串出现的次数构建一个小顶堆，然后接着遍历 hash_map，只要遍历到的字符串出现次数大于堆顶字符串的出现次数，就用遍历的字符串替换堆顶的字符串，然后再调整为小顶堆。

3）对所有剩余的字符串都遍历一次，遍历完成后堆中的 10 个字符串就是出现次数最多的字符串。这一步的时间复杂度为 O(Nlog10)。

方法三：trie 树法

方法二中使用 hash_map 来统计每个字符串出现的次数。当这些字符串有大量相同前缀时，可以考虑使用 trie 树来统计字符串出现的次数。在树的结点中保存字符串出现的次数，0 表示没有出现。具体的实现方法：在遍历的时候，从 trie 树中查找，如果找到，则把结点中保存的字符串出现次数加 1；否则为这个字符串构建新的结点，构建完成后把叶子结点中字符串的出现次数设置为 1。这样遍历完字符串后就可以知道每个字符串的出现次数，然后通过遍历这个树就可以找出出现次数最多的字符串。

trie 树经常被用来统计字符串的出现次数。它的另外一个大的用途就是字符串查找，判断是否有重复的字符串等。

10.7 如何统计不同电话号码的个数

难度系数：★★★★☆ **被考查系数：★★★★☆**

题目描述：

已知某个文件内包含一些电话号码，每个号码为 8 位数字，统计不同号码的个数。

分析与解答：

这个题目从本质上而言也是求解数据重复的问题，对于这类问题，首先会考虑位图法。对本题而言，8 位电话号码可以表示的范围是：0000 0000～9999 9999。如果用 1 个位表示一个号码，则总共需要 1 亿个位，大约 100MB 的内存。

通过上面的分析可知，这道题的主要思路：申请一个位图并初始化为 0，然后遍历所有电话号码，把遍历到的电话号码对应位图中的位设置为 1。当遍历完成后，如果位值为 1 则表示这个电话号码在文件中存在，否则这个位对应的电话号码在文件中不存在。所以位值为 1 的数量就是不同电话号码的个数。

那么对于这道题而言，最核心的算法是如何确定电话号码对应的是位图中的哪一位。下面重点介绍这个转化的方法，这里使用下面的对应方法。

00000000 对应位图的最后一位：0x0000…0000001。

00000001 对应位图倒数第二位：0x0000…0000010（1 向左移 1 位）。

00000002 对应位图倒数第三位：0x0000…0000100（1 向左移 2 位）。

00000012 对应位图的倒数第十三位：0x0000…0001 0000 0000 0000。

通常而言，位图都是通过一个整数数组来实现的（这里假设一个整数占用 4 个字节）。由此可以得出，通过电话号码获取位图中对应位置的方法为（假设电话号码为 P）：

1）通过 P/32 就可以计算出该电话号码在 bitmap 数组的下标（因为每个整数占用 32 位，通过这个公式就可以确定这个电话号码需要移动多少个 32 位，也就是可以确定它对应的位在数组中的位置）。

2）通过 P%32 就可以计算出这个电话号码在这个整型数字中具体的位置，也就是 1 这个数字对应的左移次数。因此，可以通过把 1 向左移 P%32 位，然后将得到的值与这个数组中的值做或运算，这样就能把这个电话号码在位图中对应的位设置为 1。

10.8 如何从 5 亿个数中找出中位数

难度系数：★★★★☆ **被考查系数：★★★★☆**

题目描述：

从 5 亿个数中找出中位数。数据排序后，位置在最中间的数就是中位数。当样本数为奇数时，中位数=(N+1)/2；当样本数为偶数时，中位数为 N/2 与 1+N/2 的均值。

分析与解答：

如果这道题目没有内存大小的限制，则可以把所有的数字排序后找出中位数，但是最好的排序算法的时间复杂度都是 O(NlogN)（N 为数字的个数）。这里介绍另外一种求解中位数的算法——双堆法。

方法一：双堆法

这个算法的主要思路是维护两个堆，一个大顶堆和一个小顶堆，且这两个堆需要满足如下两个特性。

特性一：大顶堆中最大的数小于或等于小顶堆中最小的数。

特性一：保证这两个堆中的元素个数的差不能超过 1。

当数据总数为偶数时，在这两个堆建立好以后，中位数显然就是两个堆顶元素的平均值。当数据总数为奇数时，根据两个堆的大小，中位数一定在数据多的堆的堆顶。对本题而言，具体实现思路：维护两个堆 maxHeap 与 minHeap，这两个堆的大小分别为 max_size 和 min_size，然后开始遍历数字，对于遍历到的数字 data 有如下 3 种情况：

1）如果 data＜maxHeap 的堆顶元素，此时为了满足特性一，只能把 data 插入到 maxHeap 中。为了满足特性二，需要分以下几种情况讨论。

① 如果 max_size≤min_size，则说明大顶堆元素个数小于小顶堆元素个数，则把 data 直接插入大顶堆中，并把这个堆调整为大顶堆。

② 如果 max_size＞min_size，为了保持两个堆元素个数的差不超过 1，则需要把 maxHeap 堆顶的元素移动到 minHeap 中，接着把 data 插入到 maxHeap 中。同时通过对堆的调整分别让两个堆保持大顶堆与小顶堆的特性。

2）如果 maxHeap 堆顶元素≤data≤minHeap 堆顶元素，则为了满足特性一，可以把 data 插入任意一个堆中。为了满足特性二，需要分以下几种情况讨论：

① 如果 max_size＜min_size，显然需要把 data 插入到 maxHeap 中；

② 如果 max_size＞min_size，显然需要把 data 插入到 minHeap 中；

③ 如果 max_size=min_size，可以把 data 插入到任意一个堆中。

3）如果 data＞maxHeap 的堆顶元素，此时为了满足特性一，只能把 data 插入到 minHeap 中。为了满足特性二，需要分以下几种情况讨论。

① 如果 max_size≥min_size，那么把 data 插入到 minHeap 中。

② 如果 max_size＜min_size，那么需要把 minHeap 堆顶元素移到 maxHeap 中，然后把 data 插入到 minHeap 中。

通过上述方法可以把 5 亿个数构建两个堆，两个堆顶元素的平均值就是中位数。

这种方法需要把所有的数据都加载到内存中，当数据量很大时，由于无法把数据一次性加载到内存中，因此这种方法比较适用于数据量小的情况。对本题而言，5 亿个数字，每个数字在内存中占 4GB，5 亿个数字需要的内存空间为 2GB 内存。如果可用的内存不足 2GB 时，则显然不能使用这种方法，因此下面介绍另外一种方法。

方法二：分治法

分治法的核心思想为把一个大的问题逐渐转换为规模较小的问题来求解。对本题而言，顺序读取这 5 亿个数字；

1）对于读取到的数字 num，如果它对应的二进制中最高位为 1，则把这个数字写入到 f1 中，如果最高位是 0，则写入到 f0 中。通过这一步就可以把这 5 亿个数字划分成了两部分，而且 f0 中的数字都大于 f1 中的数字（因为最高位是符号位）。

2）通过上面的划分可以非常容易地知道中位数是在 f0 中还是在 f1 中。假设 f1 中有 1 亿个数，那么中位数一定在文件 f0 中，从小到大第 1.5 亿个数与它后面的一个数求平均值。

3）对于 f0 可以用次高位的二进制的值继续把这个文件一分为二，使用同样的思路可以确定中位数是文件中的第几个数。直到划分后的文件可以被加载到内存中为止，接着把数据加载到内存后再进行排序，从而找出中位数。

需要注意的是，这里有一种特殊情况需要考虑，当数据总数为偶数时，如果把文件一分为二后发现两个文件中的数据有相同的个数，那么中位数就是数据比较小的文件中的最大值与数据比较大的文件中的最小值的平均值。对于求一个文件中所有数据的最大值或最小值，可以使用前面介绍的分治法进行求解。

10.9 如何按照 query 的频度排序

难度系数：★★★★☆　　被考查系数：★★★★★

题目描述：

有 10 个文件，每个文件大小 1GB，每个文件的每一行存放的都是用户的 query，每个文件的 query 都可能重复。要求按照 query 的频度排序。

分析与解答：

对于这种题，如果 query 的重复度比较大，则可以考虑一次性把所有的 query 读入到内存中处理；如果 query 的重复率不高，可用的内存不足以容纳所有的 query，那么就需要使用分治法或者其他的方法来解决。

方法一：hash_map 法

如果 query 的重复率比较高，则说明不同的 query 总数比较小，可以考虑把所有的 query 都加载到内存中的 hash_map 中（由于 hash_map 中针对每个不同的 query 只保存一个键值对，因此这些 query 占用的空间会远小于 10GB，有希望把它们一次性都加载到内存中）。接着就可以对 hash_map 按照 query 出现的次数进行排序。

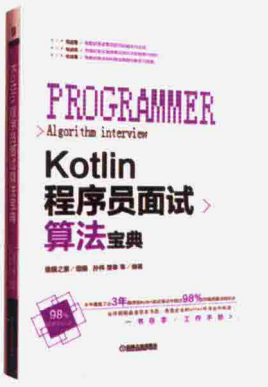
方法二：分治法

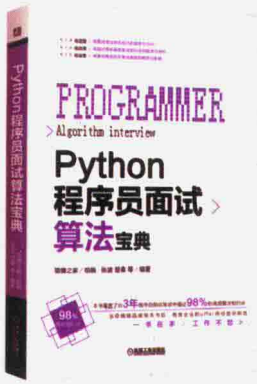
这种方法需要根据数据量的大小以及可用内存的大小来确定问题划分的规模。对本题而言，可以顺序遍历 10 个文件中的 query，通过 hash()函数执行 hash(query)%10 把这些 query 划分到 10 个文件中，这样划分后，每个文件的大小都为 1GB 左右。当然也可以根据实际情况来调整 hash()函数。如果可用内存很小，则可以把这些 query 划分到更多更小的文件中。

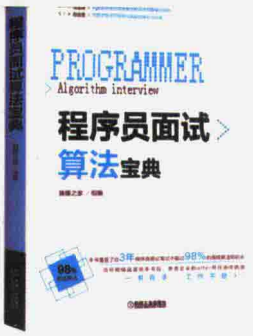
如果划分后的文件还是比较大，则可以使用相同的方法继续划分，直到每个文件都可以被读取到内存中进行处理为止，然后对每个划分后的小文件使用 hash_map 统计每个 query 出现的次数，最后再根据出现次数排序，并把排序好的 query 以及出现次数写入到另外一个单独的文件中。这样针对每个文件，都可以得到一个按照 query 出现次数排序的文件。

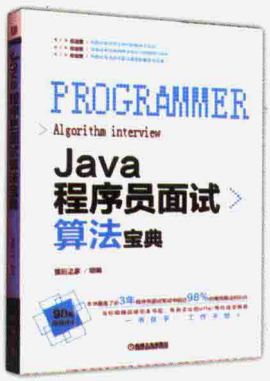
最后对所有的文件按照 query 的出现次数进行排序，这里可以使用归并排序（由于无法把所有的 query 都读入到内存中，因此这里需要使用外排序）。

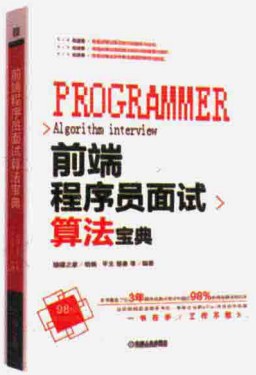
程序员算法宝典系列推荐











机械工业出版社
微信服务号

IT有得聊

上架指导 计算机/程序设计
ISBN:978-7-111-62539-1

ISBN 978-7-111-62539-1
9 787111 625391 >
定价：59.00元