十七道海量数据处理面试题与 Bit-map 详解

作者：小桥流水，redfox66，July。

前言

本博客内曾经整理过有关海量数据处理的 10 道面试题（十道海量数据处理面试题与十个方法大总结），此次除了重复了之前的 10 道面试题之后，重新多整理了 7 道。仅作各位参考，不作它用。

同时，程序员编程艺术系列将重新开始创作，第十一章以后的部分题目来源将取自下文中的 17 道海量数据处理的面试题。因为，我们觉得，下文的每一道面试题都值得重新思考，重新深究与学习。再者，编程艺术系列的前十章也是这么来的。若您有任何问题或建议，欢迎不吝指正。谢谢。

第一部分、十五道海量数据处理面试题

1. 给定 a、b 两个文件，各存放 50 亿个 url，每个 url 各占 64 字节，内存限制是 4G，让你找出 a、b 文件共同的 url？

方案 1：可以估计每个文件安的大小为 50G×64=320G，远远大于内存限制的 4G。所以不可能将其完全加载到内存中处理。考虑采取分而治之的方法。

1. 遍历文件 a，对每个 url 求取$\text{hash}(url)\%1000$，然后根据所取得的值将 url 分别存储到 1000 个小文件（记为$a_0, a_2, \cdots, a_{999}$,这里漏写个了 a1）中。这样每个小文件的大约为 300M。
2. 遍历文件 b，采取和 a 相同的方式将 url 分别存储到 1000 小文件中（记为$b_0, b_1, \cdots, b_{999}$）。这样处理后，所有可能相同的 url 都在对应的小文件（a_0 vs b_0, a_1 vs $b_1, \ldots, a_0$ vs b_{999}）中，不对应的小文件不可能有相同的 url。然后我们只要求出 1000 对小文件中相同的 url 即可。

3. 求每对小文件中相同的 url 时，可以把其中一个小文件的 url 存储到 hash_set 中。然后遍历另一个小文件的每个 url，看其是否在刚才构建的 hash_set 中，如果是，那么就是共同的 url，存到文件里面就可以了。

方案 2：如果允许有一定的错误率，可以使用 Bloom filter，4G 内存大概可以表示 340 亿 bit。将其中一个文件中的 url 使用 Bloom filter 映射为这 340 亿 bit，然后挨个读取另外一个文件的 url，检查是否与 Bloom filter，如果是，那么该 url 应该是共同的 url（注意会有一定的错误率）。

读者反馈@crowgns：

1. hash 后要判断每个文件大小，如果 hash 分的不均衡有文件较大，还应继续 hash 分文件，换个 hash 算法第二次再分较大的文件，一直分到没有较大的文件为止。这样文件标号可以用 A1-2 表示（第一次 hash 编号为 1，文件较大所以参加第二次 hash，编号为 2）
2. 由于 1 存在，第一次 hash 如果有大文件，不能用直接 set 的方法。建议对每个文件都先用字符串自然顺序排序，然后具有相同 hash 编号的（如都是 1-3，而不能 a 编号是 1，b 编号是 1-1 和 1-2），可以直接从头到尾比较一遍。对于层级不一致的，如 a1，b 有 1-1，1-2-1，1-2-2，层级浅的要和层级深的每个文件都比较一次，才能确认每个相同的 uri。

2. 有 10 个文件，每个文件 1G，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。要求你按照 query 的频度排序。

方案 1：

1. 顺序读取 10 个文件，按照 hash(query)%10 的结果将 query 写入到另外 10 个文件（记为 $a_0, a_2, \cdots, a_9$ ）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。
2. 找一台内存在 2G 左右的机器，依次对$a_0, a_2, \cdots, a_9$用 hash_map(query, query_count)来统计每个 query 出现的次数。利用快速/堆/归并排序按照出现次数进行排序。将排序好的 query 和对应的 query_cout 输出到文件中。这样得到了 10 个排好序的文件（$b_0, b_1, \cdots, b_{10}$,此处有误，更正为 b0,b1,b2,b9）。
3. 对$b_0, b_1, \cdots, b_{10}$这 10 个文件进行归并排序（内排序与外排序相结合）。

方案 2：

一般 query 的总量是有限的，只是重复的次数比较多而已，可能对于所有的 query，一次性就可以加入到内存了。这样，我们就可以采用 trie 树/hash_map 等直接来统计每个 query 出现的次数，然后按出现次数做快速/堆/归并排序就可以了

（**读者反馈**@店小二：原文第二个例子中：“找一台内存在 2G 左右的机器，依次对用 hash_map(query, query_count)来统计每个 query 出现的次数。” 由于 query 会重复，作为 key 的话，应该使用 hash_multimap 。hash_map 不允许 key 重复。@hywangw:店小二所述的肯定是错的，hash_map(query,query_count)是用来统计每个 query 的出现次数 又不是存储他们的值 出现一次 把 count+1 就行了 用 multimap 干什么？多谢 hywangw）。

方案 3：

与方案 1 类似，但在做完 hash，分成多个文件后，可以交给多个文件来处理，采用分布式的架构来处理（比如 MapReduce），最后再进行合并。

3. 有一个 1G 大小的一个文件，里面每一行是一个词，词的大小不超过 16 字节，内存限制大小是 1M。返回频数最高的 100 个词。

方案 1：顺序读文件中，对于每个词 x，取$\text{hash}(x)\%5000$，然后按照该值存到 5000 个小文件（记为$x_0, x_1, \cdots, x_{4999}$）中。这样每个文件大概是 200k 左右。如果其中的有的文件超过了 1M 大小，还可以按照类似的方法继续往下分，直到分解得到的小文件的大小都不超过 1M。对每个小文件，统计每个文件中出现的词以及相应的频率（可以采用 trie 树/hash_map 等），并取出出现频率最大的 100 个词（可以用含 100 个结点的最小堆），并把 100 词及相应的频率存入文件，这样又得到了 5000 个文件。下一步就是把这 5000 个文件进行归并（类似与归并排序）的过程了。

4. 海量日志数据，提取出某日访问百度次数最多的那个 IP。

方案 1：首先是这一天，并且是访问百度的日志中的 IP 取出来，逐个写入到一个大文件中。注意到 IP 是 32 位的，最多有 2^32 个 IP。同样可以采用映射的方法，比如模 1000，把整个大文件映射为 1000 个小文件，再找出每个小文中出现频率最大的 IP（可以采用 hash_map 进行频率统计，然后再找出频率最大的几个）及相应的频率。然后再在这 1000 个最大的 IP 中，找出那个频率最大的 IP，即为所求。

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

方案 1：采用 2-Bitmap（每个数分配 2bit，00 表示不存在，01 表示出现一次，10 表示多次，11 无意义）进行，共需内存 2^32*2bit=1GB 内存，还可以接受。然后扫描这 2.5 亿个整数，查看 Bitmap 中相对应位，如果是 00 变 01，01

变 10，10 保持不变。所描完事后，查看 bitmap，把对应位是 01 的整数输出即可。

方案 2：也可采用上题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

6. 海量数据分布在 100 台电脑中，想个办法高效统计出这批数据的 TOP10。

方案 1：

1. 在每台电脑上求出 TOP10，可以采用包含 10 个元素的堆完成（TOP10 小，用最大堆，TOP10 大，用最小堆）。比如求 TOP10 大，我们首先取前 10 个元素调整成最小堆，如果发现，然后扫描后面的数据，并与堆顶元素比较，如果比堆顶元素大，那么用该元素替换堆顶，然后再调整为最小堆。最后堆中的元素就是 TOP10 大。
2. 求出每台电脑上的 TOP10 后，然后把这 100 台电脑上的 TOP10 组合起来，共 1000 个数据，再利用上面类似的方法求出 TOP10 就可以了。

（更多可以参考：**第三章、寻找最小的 k 个数**，以及第三章续、**Top K 算法问题的实现**）

读者反馈@QinLeopard：

第 6 题的方法中，是不是不能保证每个电脑上的前十条，肯定包含最后频率最高的前十条呢？

比如说第一个文件中：A(4), B(5), C(6), D(3)

第二个文件中：A(4),B(5),C(3),D(6)

第三个文件中: A(6), B(5), C(4), D(3)

如果要选 Top(1)，选出来的结果是 A，但结果应该是 B。

@July：我想，这位读者可能没有明确提议。本题目中的 **TOP10 是指最大的 10 个数**，而不是指出现频率最多的 10 个数。但如果说，现在有另外一提，要你

求频率最多的 10 个，相当于求访问次数最多的 10 个 IP 地址那道题，即是本文中上面的第 ***4*** 题。特此说明。

7. 怎么在海量数据中找出重复次数最多的一个？

方案 1：先做 hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）。

8. 上千万或上亿数据（有重复），统计其中出现次数最多的钱 N 个数据。

方案 1：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用 hash_map/搜索二叉树/红黑树等来进行统计次数。然后就是取出前 N 个出现次数最多的数据了，可以用第 6 题提到的堆机制完成。

9. 1000 万字符串，其中有些是重复的，需要把重复的全部去掉，保留没有重复的字符串。请怎么设计和实现？

方案 1：这题用 trie 树比较合适，hash_map 也应该能行。

10. 一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词，请给出思想，给出时间复杂度分析。

方案 1：这题是考虑时间效率。用 trie 树统计每个词出现的次数，时间复杂度是 O(n*le)（le 表示单词的平准长度）。然后是找出出现最频繁的前 10 个词，可以用堆来实现，前面的题中已经讲到了，时间复杂度是 O(n*lg10)。所以总的时间复杂度，是 O(n*le)与 O(n*lg10)中较大的哪一个。

11. 一个文本文件，找出前 10 个经常出现的词，但这次文件比较长，说是上亿行或十亿行，总之无法一次读入内存，问最优解。

方案 1：首先根据用 hash 并求模，将文件分解为多个小文件，对于单个文件利用上题的方法求出每个文件件中 10 个最常出现的词。然后再进行归并处理，找出最终的 10 个最常出现的词。

12. 100w 个数中找出最大的 100 个数。

- 方案 1：采用局部淘汰法。选取前 100 个元素，并排序，记为序列 L。然后一次扫描剩余的元素 x，与排好序的 100 个元素中最小的元素比，如果比这个最小的要大，那么把这个最小的元素删除，并把 x 利用插入排序的思想，插入到序列 L 中。依次循环，知道扫描了所有的元素。复杂度为 O(100w*100)。
- 方案 2：采用快速排序的思想，每次分割之后只考虑比轴大的一部分，知道比轴大的一部分在比 100 多的时候，采用传统排序算法排序，取前 100 个。复杂度为 O(100w*100)。
- 方案 3：在前面的题中，我们已经提到了，用一个含 100 个元素的最小堆完成。复杂度为 O(100w*lg100)。

13. 寻找热门查询：

搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为 1-255 字节。假设目前有一千万个记录，这些查询串的重复读比较高，虽然总数是 1 千万，但是如果去除重复和，不超过 3 百万个。一个查询串的重复度越高，说明查询它的用户越多，也就越热门。请你统计最热门的 10 个查询串，要求使用的内存不能超过 1G。

(1) 请描述你解决这个问题的思路；

(2) 请给出主要的处理流程，算法，以及算法的复杂度。

方案 1：采用 trie 树，关键字域存该查询串出现的次数，没有出现为 0。最后用 10 个元素的最小推来对出现频率进行排序。

关于此问题的详细解答，请参考此文的第 3.1 节：**第三章续、Top K 算法问题的实现**。

14. 一共有 N 个机器，每个机器上有 N 个数。每个机器最多存 O(N)个数并对它们操作。如何找到 N^2 个数中的中数？

方案 1：先大体估计一下这些数的范围，比如这里假设这些数都是 32 位无符号整数（共有 2^32 个）。我们把 0 到 2^32-1 的整数划分为 N 个范围段，每个段包含（2^32）/N 个整数。比如，第一个段位 0 到 2^32/N-1，第二段为（2^32）/N 到（2^32）/N-1，...，第 N 个段为（2^32）（N-1）/N 到 2^32-1。然后，扫描每个机器上的 N 个数，把属于第一个区段的数放到第一个机器上，属于第二个区段的数放到第二个机器上，...，属于第 N 个区段的数放到第 N 个机器上。注意这个过程每个机器上存储的数应该是 O(N)的。下面我们依次统计每个机器上数的个数，一次累加，直到找到第 k 个机器，在该机器上累加的数大于或等于（N^2）/2，而在第 k-1 个机器上的累加数小于（N^2）/2，并把这个数记为 x。那么我们要找的中位数在第 k 个机器中，排在第（N^2）/2-x 位。然后我们对第 k 个机器的数排序，并找出第（N^2）/2-x 个数，即为所求的中位数的复杂度是 O（N^2）的。

方案 2：先对每台机器上的数进行排序。排好序后，我们采用归并排序的思想，将这 N 个机器上的数归并起来得到最终的排序。找到第（N^2）/2 个便是所求。复杂度是 O（N^2*lgN^2）的。

15. 最大间隙问题

给定 n 个实数$x_1, x_2, x_3, \cdots, x_n$，求着 n 个实数在实轴上向量 2 个数之间的最大差值，要求线性的时间算法。

方案 1：最先想到的方法就是先对这 n 个数据进行排序，然后一遍扫描即可确定相邻的最大间隙。但该方法不能满足线性时间的要求。故采取如下方法：

1. 找到 n 个数据中最大和最小数据 max 和 min。
2. 用 n-2 个点等分区间[min, max]，即将[min, max]等分为 n-1 个区间（前闭后开区间），将这些区间看作桶，编号为$1, 2, \cdots, n-2, n-1$，且桶 i 的上界和桶 i+1 的下届相同，即每个桶的大小相同。每个桶的大小为：$\text{dblAvrGap} = \frac{(max-min)}{n-1}$。实际上，这些桶的边界构成了一个等差数列（首项为 min，公差为$d = \text{dblAvrGap}$），且认为将 min 放入第一个桶，将 max 放入第 n-1 个桶。
3. 将 n 个数放入 n-1 个桶中：将每个元素 x[i] 分配到某个桶（编号为 index），其中 $\text{index} = \left\lfloor \frac{(x[i]-\min)}{dblAvrGap} + \right\rfloor + 1$（这括号里多了个“+”），并求出分到每个桶的最大最小数据。
4. 最大间隙：除最大最小数据 max 和 min 以外的 n-2 个数据放入 n-1 个桶中，由抽屉原理可知至少有一个桶是空的，又因为每个桶的大小相同，所以最大间隙不会在同一桶中出现，一定是某个桶的上界和气候某个桶的下界之间隙，且该量筒之间的桶（即便好在该连个便好之间的桶）一定是空桶。也就是说，最大间隙在桶 i 的上界和桶 j 的下界之间产生 j>=i+1。一遍扫描即可完成。

16. 将多个集合合并成没有交集的集合

给定一个字符串的集合，格式如：$\{aaa, bbb, ccc\}, \{bbb, ddd\}, \{eee, fff\}, \{ggg\}, \{ddd, hhh\}$。要求将其中交集不为空的集合合并，要求合并完成的集合之间无交集，例如上例应输出$\{aaa, bbb, ccc, ddd, hhh\}, \{eee, fff\}, \{ggg\}$。

(1) 请描述你解决这个问题的思路；

(2) 给出主要的处理流程，算法，以及算法的复杂度；

(3) 请描述可能的改进。

方案 1：采用并查集。首先所有的字符串都在单独的并查集中。然后依扫描每个集合，顺序合并将两个相邻元素合并。例如，对于$\{aaa, bbb, ccc\}$，首先查看 aaa 和 bbb 是否在同一个并查集中，如果不在，那么把它们所在的并查集合并，然后再看 bbb 和 ccc 是否在同一个并查集中，如果不在，那么也把它们所在的并查集合并。接下来再扫描其他的集合，当所有的集合都扫描完了，并查集代表的集合便是所求。复杂度应该是 O(NlgN)的。改进的话，首先可以记录每个节点的根结点，改进查询。合并的时候，可以把大的和小的进行合，这样也减少复杂度。

17. 最大子序列与最大子矩阵问题

数组的最大子序列问题：给定一个数组，其中元素有正，也有负，找出其中一个连续子序列，使和最大。

方案 1：这个问题可以动态规划的思想解决。设 b[i]表示以第 i 个元素 a[i]结尾的最大子序列，那么显然$b[i+1] = b[i] > 0 ? b[i] + a[i+1] : a[i+1]$。基于这一点可以很快用代码实现。

最大子矩阵问题：给定一个矩阵（二维数组），其中数据有大有小，请找一个子矩阵，使得子矩阵的和最大，并输出这个和。

方案 2：可以采用与最大子序列类似的思想来解决。如果我们确定了选择第 i 列和第 j 列之间的元素，那么在这个范围内，其实就是一个最大子序列问题。如何确定第 i 列和第 j 列可以词用暴搜的方法进行。

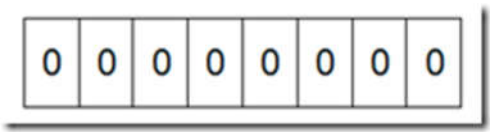
第二部分、海量数据处理之 Bti-map 详解

Bloom Filter 已在上一篇文章**海量数据处理之 Bloom Filter 详解**中予以详细阐述，本文接下来着重阐述 Bit-map。有任何问题，欢迎不吝指正。

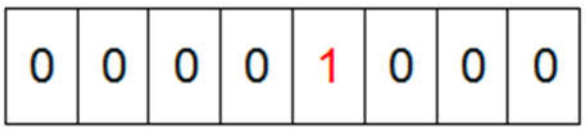
什么是 Bit-map

所谓的 Bit-map 就是用一个 bit 位来标记某个元素对应的 Value， 而 Key 即是该元素。由于采用了 Bit 为单位来存储数据，因此在存储空间方面，可以大大节省。

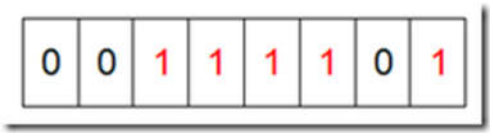
如果说了这么多还没明白什么是 Bit-map，那么我们来看一个具体的例子，假设我们要对 0-7 内的 5 个元素(4,7,2,5,3)排序（这里假设这些元素没有重复）。那么我们就可以采用 Bit-map 的方法来达到排序的目的。要表示 8 个数，我们就只需要 8 个 Bit（1Bytes），首先我们开辟 1Byte 的空间，将这些空间的所有 Bit 位都置为 0(如下图：)



然后遍历这 5 个元素，首先第一个元素是 4，那么就把 4 对应的位置为 1（可以这样操作 p+(i/8)|(0×01<<(i%8)) 当然了这里的操作涉及到 Big-ending 和 Little-ending 的情况，这里默认为 Big-ending）,因为是从零开始的，所以要把第五位置为一（如下图）：



然后再处理第二个元素 7，将第八位置为 1,，接着再处理第三个元素，一直到最后处理完所有的元素，将相应的位置为 1，这时候的内存的 Bit 位的状态如下：



然后我们现在遍历一遍 Bit 区域，将该位是一的位的编号输出（2，3，4，5，7），这样就达到了排序的目的。下面的代码给出了一个 BitMap 的用法：排序。

```
1.  //定义每个 Byte 中有 8 个 Bit 位
```

```
2.  #include <memory.h>
3.  #define BYTESIZE 8
4.  void SetBit(char *p, int posi)
5.  {
6.      for(int i=0; i < (posi/BYTESIZE); i++)
7.      {
8.          p++;
9.      }
10.
11.     *p = *p|(0x01<<(posi%BYTESIZE));//将该 Bit 位赋值 1
12.     return;
13. }
14.
15. void BitMapSortDemo()
16. {
17.     //为了简单起见，我们不考虑负数
18.     int num[] = {3,5,2,10,6,12,8,14,9};
19.
20.     //BufferLen 这个值是根据待排序的数据中最大值确定的
21.     //待排序中的最大值是 14，因此只需要 2 个 Bytes(16 个 Bit)
22.     //就可以了。
23.     const int BufferLen = 2;
24.     char *pBuffer = new char[BufferLen];
25.
26.     //要将所有的 Bit 位置为 0，否则结果不可预知。
27.     memset(pBuffer,0,BufferLen);
28.     for(int i=0;i<9;i++)
29.     {
30.         //首先将相应 Bit 位上置为 1
31.         SetBit(pBuffer,num[i]);
32.     }
33.
34.     //输出排序结果
35.     for(int i=0;i<BufferLen;i++)//每次处理一个字节(Byte)
36.     {
37.         for(int j=0;j<BYTESIZE;j++)//处理该字节中的每个 Bit 位
38.         {
39.             //判断该位上是否是 1，进行输出，这里的判断比较笨。
40.             //首先得到该第 j 位的掩码（0x01<<j），将内存区中的
41.             //位和此掩码作与操作。最后判断掩码是否和处理后的
42.             //结果相同
43.             if((*pBuffer&(0x01<<j)) == (0x01<<j))
44.             {
45.                 printf("%d ",i*BYTESIZE + j);
```

```
            }
        }
        pBuffer++;
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    BitMapSortDemo();
    return 0;
}
```

可进行数据的快速查找，判重，删除，一般来说数据范围是 int 的 10 倍以下

基本原理及要点

使用 bit 数组来表示某些元素是否存在，比如 8 位电话号码

扩展

Bloom filter 可以看做是对 bit-map 的扩展（关于 Bloom filter，请参见：海量数据处理之 **Bloom filter** 详解）。

问题实例

1)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。

8 位最多 99 999 999，大概需要 99m 个 bit，大概 10 几 m 字节的内存即可。（可以理解为从 0-99 999 999 的数字，每个数字对应一个 Bit 位，所以只需要 99M 个 Bit==1.2MBytes，这样，就用了小小的 1.2M 左右的内存表示了所有的 8 位数的电话）

2)2.5 亿个整数中找出不重复的整数的个数，内存空间不足以容纳这 2.5 亿个整数。

将 bit-map 扩展一下，用 2bit 表示一个数即可，0 表示未出现，1 表示出现一次，2 表示出现 2 次及以上，在遍历这些数的时候，如果对应位置的值是 0，则将其置为 1；如果是 1，将其置为 2；如果是 2，则保持不变。或者我们不用 2bit 来进行表示，我们用两个 bit-map 即可模拟实现这个 2bit-map，都是一样的道理。

参考：

1. http://www.cnblogs.com/youwang/archive/2010/07/20/1781431.html。
2. http://blog.redfox66.com/post/2010/09/26/mass-data-4-bitmap.aspx。

完。