**set/map/multiset/multimap(内部：红黑树)**

set，同map一样，所有元素都会根据元素的键值自动被排序，因为set/map两者的所有各种操作，都只是转而调用RB-tree的操作行为，不过，值得注意的是，两者都不允许两个元素有相同的键值。  
    不同的是：set的元素不像map那样可以同时拥有实值(value)和键值(key)，set元素的键值就是实值，实值就是键值，而map的所有元素都是pair，同时拥有实值(value)和键值(key)，pair的第一个元素被视为键值，第二个元素被视为实值。  
    至于multiset/multimap，他们的特性及用法和set/map完全相同，唯一的差别就在于它们允许键值重复，即所有的插入操作基于RB-tree的insert\_equal()而非insert\_unique()。

**hash\_set/hash\_map/hash\_multiset/hash\_multimap（内部：hashtable）**

    hash\_set/hash\_map，两者的一切操作都是基于hashtable之上。不同的是，hash\_set同set一样，同时拥有实值和键值，且实质就是键值，键值就是实值，而hash\_map同map一样，每一个元素同时拥有一个实值(value)和一个键值(key)，所以其使用方式，和上面的map基本相同。但由于hash\_set/hash\_map都是基于hashtable之上，所以不具备自动排序功能。为什么?因为hashtable没有自动排序功能。  
    至于hash\_multiset/hash\_multimap的特性与上面的multiset/multimap完全相同，唯一的差别就是它们hash\_multiset/hash\_multimap的底层实现机制是hashtable（而multiset/multimap，上面说了，底层实现机制是RB-tree），所以它们的元素都不会被自动排序，不过也都允许键值重复。

### 方法一：分而治之/Hash映射 + Hash\_map统计 + 堆/快速/归并排序

**1、海量日志数据，提取出某日访问百度次数最多的那个IP。**

1. 分而治之/hash映射：针对数据太大，内存受限，只能是：把大文件化成(取模映射)小文件，即16字方针：大而化小，各个击破，缩小规模，逐个解决
2. hash\_map统计：当大文件转化了小文件，那么我们便可以采用常规的hash\_map(ip，value)来进行频率统计。Ip为键，出现的次数为值
3. 堆/快速排序：统计完了之后，便进行排序(可采取堆排序)，得到次数最多的IP。

具体做法：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，最多有个2^32个IP。同样可以采用映射的方法，比如%1000，把整个大文件映射为1000个小文件，再找出每个小文件中出现频率最大的IP（可以采用hash\_map对那1000个文件中的所有IP进行频率统计，然后依次找出各个文件中频率最大的那个IP）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。

**2、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。**

1. 分而治之/hash映射：顺序读文件中，对于每个词x，取hash(x)%5000，然后按照该值存到5000个小文件（记为x0,x1,...x4999）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。
2. hash\_map统计：对每个小文件，采用trie树/hash\_map等统计每个文件中出现的词以及相应的频率。
3. 堆/归并排序：取出出现频率最大的100个词（可以用含100个结点的最小堆）后，再把100个词及相应的频率存入文件，这样又得到了5000个文件。最后就是把这5000个文件进行归并（类似于归并排序）的过程了。

### 方法二：Bloom filter/Bitmap

**1、在2.5亿个整数中找出不重复的整数，注，内存不足以容纳这2.5亿个整数。**

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 \* 2 bit=1 GB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。

**2、给40亿个不重复的unsigned int的整数，没排过序的，然后再给一个数，如何快速判断这个数是否在那40亿个数当中？**

1个unsigned int占用4字节，40亿大约是4G个数，那么一共大约要用16G的内存空间。**1个unsigned int是一个2^32以内的数，一个数可以表示32个数是否存在。**理申请一个unsigned int的数组a[n]则可以表示连续的 n\*32的数。也就是a[0]表示0-31的数是否存在，a[1]表示32-63的数是否存在，依次类推。此时需要的内存：16G/32=512M。利用1bit位代表一个unsigned int值，读入40亿个数，设置相应的bit位u，读入要查询的数，查看相应bit位是否为1，为1表示存在，为0表示不存在。

### 方法三、分布式处理之Mapreduce

MapReduce是一个编程模型，也是一个处理和生成超大数据集的算法模型的相关实现。用户首先创建一个Map函数处理一个基于key/value pair的数据集合，输出中间的基于key/value pair的数据集合；然后再创建一个Reduce函数用来合并所有的具有相同中间key值的中间value值。

### 方法四、双层桶划分

**1.文件中有 10G 个整数，乱序排列，要求找出中位数。内存限制为 2G。只写出思路即可（内存限制为 2G的意思就是，可以使用2G的空间来运行程序，而不考虑这台机器上的其他软件的占用内存）。**5亿个int找它们的中位数。（类似）

**思想：将整形的每1byte作为一个关键字，也就是说一个整形可以拆成4个keys，而且最高位的keys越大，整数越大。如果高位keys相同，则比较次高位的keys。整个比较过程类似于字符串的字典序。**

**第一步:把10G整数每2G读入一次内存，然后一次遍历这536,870,912个数据。每个数据用位运算”>>”取出最高8位(31-24)。这8bits(0-255)最多表示255个桶，那么可以根据8bit的值来确定丢入第几个桶。最后把每个桶写入一个磁盘文件中，同时在内存中统计每个桶内数据的数量，自然这个数量只需要255个整形空间即可。**

**第二步：根据内存中255个桶内的数量，计算中位数在第几个桶中。很显然，2,684,354,560个数中位数是第1,342,177,280个。假设前127个桶的数据量相加，发现少于1,342,177,280，把第128个桶数据量加上，大于1,342,177,280。说明，中位数必在磁盘的第128个桶中。而且在这个桶的第1,342,177,280-N(0-127)个数位上。N(0-127)表示前127个桶的数据量之和。然后把第128个文件中的整数读入内存。(平均而言，每个文件的大小估计在10G/128=80M左右，当然也不一定，但是超过2G的可能性很小)。**

**第三步：继续以内存中的整数的次高8bit进行桶排序(23-16)。过程和第一步相同，也是255个桶。   
第四步：一直下去，直到最低字节(7-0bit)的桶排序结束。我相信这个时候完全可以在内存中使用一次快排就可以了。**

#### Bloom filter

适用范围：可以用来实现数据字典，进行数据的判重，或者集合求交集   
扩展：Bloom filter将集合中的元素映射到位数组中，用k（k为哈希函数个数）个映射位是否全1表示元素在不在这个集合中。Counting bloom filter（CBF）将位数组中的每一位扩展为一个counter，从而支持了元素的删除操作。Spectral Bloom Filter（SBF）将其与集合元素的出现次数关联。SBF采用counter中的最小值来近似表示元素的出现频率。

问题实例：给你A,B两个文件，各存放50亿条URL，每条URL占用64字节，内存限制是4G，让你找出A,B文件共同的URL。如果是三个乃至n个文件呢？

方案1：根据这个问题我们来计算下内存的占用，4G=2^32大概是40亿\*8大概是340亿，n=50亿，如果按出错率0.01算需要的大概是650亿个bit。现在可用的是340亿，相差并不多，这样可能会使出错率上升些。另外如果这些url ip是一一对应的，就可以转换成ip，则大大简单了。（注意会有一定的错误率）

方案2：分而治之/hash映射：hash（url）%1000,每个大文件被分成1000个小文件，不对应的小文件不可能有相同的url；hash\_set统计：把其中一个文件中的url存储到hash\_set中，遍历另一个小文件的每个url，看是否在其中。

#### Hash

适用范围：快速查找，删除的基本数据结构，通常需要总数据量可以放入内存   
问题实例：   
1).海量日志数据，提取出某日访问百度次数最多的那个IP。   
IP的数目还是有限的，最多2^32个，所以可以考虑使用hash将ip直接存入内存，然后进行统计（分而治之/hash映射；hash\_map统计；堆/快速排序）

#### 堆

适用范围：海量数据前n大，并且n比较小，堆可以放入内存   
问题实例：   
1)100w个数中找最大的前100个数。   
用一个100个元素大小的最小堆即可。

#### 双层桶划分—-其实本质上就是【分而治之】的思想，重在“分”的技巧上

适用范围：第k大，中位数，不重复或重复的数字   
问题实例：   
1).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。   
　　有点像鸽巢原理，整数个数为2^32,也就是，我们可以将这2^32个数，划分为2^8个区域(比如用单个文件代表一个区域)，然后将数据分离到不同的区域，然后不同的区域在利用bitmap就可以直接解决了。也就是说只要有足够的磁盘空间，就可以很方便的解决。具体方案：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 \* 2 bit=1 GB内存，还可以接受（为什么这里取2^32，这是因为我们假设了最大的数就是2^32，实际上，这也是各大语言中默认的int类型最大的数。所以这2.5亿个数的取值范围必须在 0< 2^32之间）。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。我觉得思路倒是很清晰。

2).5亿个int找它们的中位数。   
　　这个例子比上面那个更明显。首先我们将int划分为2^16个区域，然后读取数据统计落到各个区域里的数的个数，之后我们根据统计结果就可以判断中位数落到那个区域，同时知道这个区域中的第几大数刚好是中位数。然后第二次扫描我们只统计落在这个区域中的那些数就可以了。

　　实际上，如果不是int是int64，我们可以经过3次这样的划分即可降低到可以接受的程度。即可以先将int64分成2^24个区域，然后确定区域的第几大数，在将该区域分成2^20个子区域，然后确定是子区域的第几大数，然后子区域里的数的个数只有2^20，就可以直接利用direct addr table进行统计了。

#### 位图

适用范围：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   
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。

每个数字用相应的位来表示：例00000001->0x00….0010，00000004->0x00…10000  
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。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。

#### 数据库索引

适用范围：大数据量的增删改查

* 关于数据库索引及其优化，更多可参见此文：<http://www.cnblogs.com/pkuoliver/archive/2011/08/17/mass-data-topic-7-index-and-optimize.html>；
* 关于MySQL索引背后的数据结构及算法原理，这里还有一篇很好的文章：<http://blog.codinglabs.org/articles/theory-of-mysql-index.html>；
* 关于B 树、B+ 树、B\* 树及R 树，有篇绝佳文章：<http://blog.csdn.net/v_JULY_v/article/details/6530142>。

#### 倒排索引

适用范围：搜索引擎，关键字查询   
扩展：   
问题实例：文档检索系统，查询那些文件包含了某单词，比如常见的学术论文的关键字搜索。   
关于倒排索引的应用，更多请参见：

* <http://blog.csdn.net/v_july_v/article/details/7085669>
* <http://blog.csdn.net/v_july_v/article/details/7109500>

#### 外排序

适用范围：大数据的排序，去重   
基本原理及要点：外排序的归并方法，置换选择败者树原理，最优归并树   
问题实例：   
1).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个字节，内存限制大小是1M。返回频数最高的100个词。   
这个数据具有很明显的特点，词的大小为16个字节，但是内存只有1m做hash有些不够，所以可以用来排序。内存可以当输入缓冲区使用。   
更多参考：   
+ <http://blog.csdn.net/v_JULY_v/article/details/6451990>

#### trie树

适用范围：数据量大，重复多，但是数据种类小可以放入内存 。用于统计和排序大量的字符串（空间换时间），建立trie的复杂度为O(n\*len)，查询O(len)  
问题实例：   
　　1).有10个文件，每个文件1G，每个文件的每一行都存放的是用户的query，每个文件的query都可能重复。要你按照query的频度排序。

本题不再是选出top N的问题。而是排序的问题，但是核心还是不变。  
1.分而治之/hash映射：由于没有说相同的query是放在同一个文件，所以必须先对每一个query做一个映射，将相同的query放在同一个文件。所以注意这里会产生10个新的文件（相同query在同一个文件）。  
2.hash统计：主要选一个2g的电脑来对每一个文件完成这个过程。因为每个文件的容量都为1g。

3.堆/快速/归并排序：先利用堆/快速/归并排序对每一份文件的query和query\_count进行排序，然后将排好序的内容再输出到一份新文件中。最后对这10个文件进行归并排序  
　　2).1000万字符串，其中有些是相同的(重复),需要把重复的全部去掉，保留没有重复的字符串。请问怎么设计和实现？   
　　3).寻找热门查询：查询串的重复度比较高，虽然总数是1千万，但如果除去重复后，不超过3百万个，每个不超过255字节。

4）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。  
解答：  
除了使用前面的分而治之/hash映射 + hash统计 + 堆/快速/归并排序的方法以外，也可以使用单词查找树。下面是单词查找树的分析：  
这题是考虑时间效率。用trie树(单词查找树,字典树，就是一种树有插入、查找，删除等操作)统计每个词出现的次数，时间复杂度是O(n\*le)（le表示单词的字符串长度）。因为在面对每一个单词的时候，我们都可以在字典书中进行查找，查看是否有这个单词，然后将相应的次数加一，之后就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。

#### 分布式处理 mapreduce

适用范围：数据量大，但是数据种类小可以放入内存   
问题实例：   
　　1).The canonical example application of MapReduce is a process to count the appearances of   
each different word in a set of documents:   
　　2).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。   
　　3).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个数的中数(median)？