IaaS 【infrastructure as a service】 基础架构即服务 Amazon AWS

SaaS 【software as a service】 Salesforce.com

PaaS 【platform as a service】 google appp engine

**Hashing**

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   
基本原理及要点： hash函数选择，针对字符串，整数，排列，具体相应的hash方法。   
碰撞处理，一种是open hashing，也称为拉链法；另一种就是closed hashing，也称开地址法，opened addressing。

数组的特点是：寻址容易，插入和删除困难；而链表的特点是：寻址困难，插入和删除容易。那么我们能不能综合两者的特性，做出一种寻址容易，插入删除也容易的数据结构？答案是肯定的，这就是我们要提起的哈希表，哈希表有多种不同的实现方法，最常用的一种方法——拉链法，我们可以理解为“链表的数组”

**bit-map**

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   
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

**双层桶划分**

事实上，与其说双层桶划分是一种数据结构，不如说它是一种算法设计思想。面对一堆大量的数据我们无法处理的时候，我们可以将其分成一个个小的单元，然后根据一定的策略来处理这些小单元，从而达到目的。

适用范围：第k大，中位数，不重复或重复的数字   
基本原理及要点：因为元素范围很大，不能利用直接寻址表，所以通过多次划分，逐步确定范围，然后最后在一个可以接受的范围内进行, 分治才是其根本。   
问题实例：

**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 （适合大数据量，求前n小，n的大小比较小的情况，这样可以扫描一遍即可得到所有的前n元素，效率很高）  
扩展：双堆，一个最大堆与一个最小堆结合，可以用来维护中位数。   
问题实例： 100w个数中找最大的前100个数。

倒排索引(Inverted index)

适用范围：搜索引擎，关键字查询   
基本原理及要点：为何叫倒排索引？一种索引方法，被用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射。   
以英文为例，下面是要被索引的文本：   
T0 = "it is what it is"   
T1 = "what is it"   
T2 = "it is a banana"   
我们就能得到下面的反向文件索引：   
"a":       {2}   
"banana": {2}   
"is":     {0, 1, 2}   
"it":     {0, 1, 2}   
"what":   {0, 1}   
检索的条件"what", "is" 和 "it" 将对应集合的交集。   
正向索引开发出来用来存储每个文档的单词的列表。正向索引的查询往往满足每个文档有序频繁的全文查询和每个单词在校验文档中的验证这样的查询。在正向索引中，文档占据了中心的位置，每个文档指向了一个它所包含的索引项的序列。也就是说文档指向了它包含的那些单词，而反向索引则是单词指向了包含它的文档，很容易看到这个反向的关系。    
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。

1. 分而治之/hash映射 + hash统计 + 堆/快速/归并排序；

2. 双层桶划分

3. Bloom filter/Bitmap；

4. Trie树/数据库/倒排索引；

5. 外排序；

6. 分布式处理之Hadoop/Mapreduce。

分而治之/Hash映射 + Hash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。**

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。

**2、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。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。），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。

第一步借用hash统计进行预处理： 先对这批海量数据预处理(维护一个Key为Query字串，Value为该Query出现次数，即Hashmap(Query，Value)，每次读取一个Query，如果该字串不在Table中，那么加入该字串，并且将Value值设为1；如果该字串在Table中，那么将该字串的计数加一即可。最终我们在O(N)（N为1千万，因为要遍历整个数组一遍才能统计处每个query出现的次数）的时间复杂度内用Hash表完成了统计；  
    第二步借用堆排序找出最热门的10个查询串：时间复杂度为N'\*logK。维护一个K(该题目中是10)大小的小根堆，然后遍历3百万个Query，分别和根元素进行对比（对比value的值），找出10个value值最大的query  
    最终的时间复杂度是：O（N） + N'\*O（logK），（N为1000万，N’为300万）

或者：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。

**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。**

第一步分而治之/hash映射到顺序读文件中，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。

第二步hash统计对每个小文件，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个词及相应的频率存入文件，这样又得到了5000个文件。  
     第三步堆/归并排序就是把这5000个文件进行归并（也可以采用堆排序）的过程了。（如果内存允许可以将这5000个文件中的所有元素合并起来，利用堆获得top 100）

**4、 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？**

　　可以估计每个文件安的大小为5G×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。

遍历文件a，对每个url求取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a0,a1,...,a999）中。这样每个小文件的大约为300M。

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（记为b0,b1,...,b999）。这样处理后，所有可能相同的url都在对应的小文件（a0vsb0,a1vsb1,...,a999vsb999）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。

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。

**位图存储（bitmap）**

**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（共有都少个不同的号码）。**

8位最多99 999 999（0-99 999 999共1亿个数），每个数字对应一个Bit位，所以只需要99MBit==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（int）中找出不重复的整数的个数，内存足够大。**

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。 （每个整数用两位，存储所有的整数需要2^32\*2=1GB的内存）

**3.  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。**

整数个数为2^32，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（占用4MB，内存可以存下）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。

**4.  5亿个int找它们的中位数 (指将统计总体当中的各个变量值按大小顺序排列起来，形成一个数列，处于变量数列中间位置的变量值就称为中位数)**

首先我们将int划分为2^16个区域（肯定是按大小的），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。

**5.  腾讯面试题：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？**

方案1：申请512M的内存（2^32/8=512MB），一个bit位代表一个unsigned int值。读入40亿个数，设置相应的bit位，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。

方案2：因为2^32为40亿多，所以给定一个数可能在，也可能不在其中；这里我们把40亿个数中的每一个用32位的二进制来表示假设这40亿个数开始放在一个文件中。

然后将这40亿个数分成两类:  1. 最高位为0  2. 最高位为1

并将这两类分别写入到两个文件中，其中一个文件中数的个数<=20亿，而另一个>=20亿（这相当于折半了）；与要查找的数的最高位比较并接着进入相应的文件再查找

再然后把这个文件为又分成两类: 1.次最高位为0 2.次最高位为1

并将这两类分别写入到两个文件中，其中一个文件中数的个数<=10亿，而另一个>=10亿（这相当于折半了）； 与要查找的数的次最高位比较并接着进入相应的文件再查找。 ....... 以此类推，就可以找到了,而且时间复杂度为O(logn)。

网络分发，自助服务，可衡量的服务，资源灵活调度，资源池化

**IaaS SaaS PaaS**

公有云，私有云，社区云，混合云

云计算的5大特征

1 自助式服务

2 通过网络分发服务

3 资源池化

4 资源灵活调度  webEx全球视频会议  amazon ec2

5 可衡量的服务

云计算安全建设：网络准入、网络加密

二层准入、三层准入、客户端准入

通过数据加密防止：

1 数据被截取

2 数据被暴露给云服务提供商

3 数据由于存储硬件的丢失而泄露

VPN virtual private network虚拟专用网络

二层准入：

802.1x supplicant客户端、authenticator认证方（交换机、无线AP或者控制器）、authentication server认证服务器。在主机利用DHCP协议获取IP地址前通过认证

认证流程：

1 端口初始化

发现有端口接入时，置为“未授权”，发送广播

2 EAP初始化  EAP extensible authentcation protocal 可扩展验证协议

接收广播，回复应答

3 EAP协商

确定接下来使用的类型

4 用户身份验证

客户机发送信息给交换机，交换机发送认证信息给认证服务器，判断是否授权

认证方与认证服务器 通常采用 RADIUS（remote authentication dial in user service 远程认证拨号接入用户服务）/TACACS+(terminal access controller access-control system plus增强型终端访问控制器访问控制系统)

三层准入：

ACL access control list 访问控制列表

1 交换机端口进入有限接入状态

2 客户端出发认证流程

3 用户身份验证

客户端方式：

安装在用户设备上的软件来验证

1 从操作系统接受802.1x的认证流程

2 对系统健康状况进行检查

二层准入：成熟、使用---完全公开的架构、成熟的技术标准、完善的认证和授权机制

三层准入：轻便、简单---零客户端、使用简单、安全性较弱、不支持single sign-on、不支持机器认证

客户端方式：功能全面、五统一标准

VPN virtual private network 虚拟个人网络：长连接和加密

L2TP(layer 2 tunneling protocol) 二层隧道协议

有编号1~100个灯泡，起初所有的灯都是灭的。有100个同学来按灯泡开关，如果灯是亮的，那么按过开关之后，灯会灭掉。如果灯是灭的，按过开关之后灯会亮。

现在开始按开关。

第1个同学，把所有的灯泡开关都按一次(按开关灯的编号： 1,2,3,……100)。

第2个同学，隔一个灯按一次(按开关灯的编号： 2,4,6,……,100)。

第3个同学，隔两个灯按一次(按开关灯的编号： 3,6,9,……,99)。

……

问题是，在第100个同学按过之后，有多少盏灯是亮着的？这些灯的编号是多少？要求给出解题思路或给出伪码。

首先，编号为n的同学会去按他编号倍数的灯，因此，想到这个是个因子分解的问题。于是就想到了质数，质数只有1和它本身两个因子，因此按两次一定是灭的。

为了让灯最后是亮的，由于最开始是灭的，因此只能按奇数次，也就是说，灯的编号要能分解成奇数个因子才行。

开始写出了从1到10的数，发现只有1,4,9有奇数个因子，其他都是偶数个因子，就猜想，是不是平方数有奇数个因子。很奇怪就研究了一下。

1 = 1 \* 1， 因子是1

4 = 1 \* 4 = 2 \* 2， 因子是1,4,2

9 = 1 \* 9 = 3 \* 3， 因子是1,9,3

16 = 1 \* 16 = 2 \* 8 = 4 \* 4， 因子是1,16,2,8,4

。。。

发现规律，每个数都能携程2个数相乘的形式，也就是说，这样就有两个因子了。为了实现有奇数个因子，必须存在相乘的两个数相同的情况。（上述的1,2,3,4）这样的话， 也就是平方数了。

综上，只有平方数编号的灯最后是亮着的。不得不说，数学实在是太美妙了！！

**1**、设m和n都是int类型，那么以下for循环语句，（ ）  
for(m=0,n=-1;n=0;m++,n++)  
{  
n++;  
}  
A:循环体一次也不执行  
B:循环体执行一次  
C:是无限循环  
D:有限次循环  
E:循环结束判断条件不合法  
F:运行出错  
  
**答案**：A  
**解析**：判断条件n=0，无论何时，n都被赋值为0，而赋值表达式的返回值即被赋值本身，即0，从而被认为是false，因此，for语句的循环体从来不会被执行。  
  
**2**、计算三个稠密矩阵A、B、C的乘积ABC，假定三个矩阵的尺寸分别为m\*n,n\*p,p\*q,且m<n<p<q，以下计算顺序效率最高的是：（ ）  
A:(AB)C  
B:A(BC)  
C:(AC)B  
D:(BC)A  
E:(CA)B  
F:以上效率相同  
  
**答案**：A  
**解析**：m\*n的矩阵A和n\*p的矩阵B的乘积，得到m\*p的矩阵A\*B，而A\*B的每个元素需要n次乘法和n-1次加法，忽略加法，共需要m\*n\*p次乘法运算。因此，A选项(AB)C需要的乘法次数是m\*n\*p+m\*p\*q；B选项A(BC)需要的乘法次数是n\*p\*q+m\*n\*q；由于m\*n\*p< m\*n\*q，m\*p\*q<n\*p\*q，显然A运算次数更少。矩阵乘法一般不满足交换律，甚至造成根本不可乘，因此C、D、E不是正确的计算方式。  
  
**3**、若干个等待访问磁盘者依次要访问的磁道为19,43,40,4,79,11,76，当前磁头位于40号柱面，若用最短寻道时间优先磁盘调度算法，则访问序列为？（ ）  
A:19,43,40,4,79,11,76  
B:40,43,19,11,4,76,79  
C:40,43,76,79,19,11,4  
D:40,43,76,79,4,11,19  
E:40,43,76,79,11,4,19  
F:40,19,11,4,79,76,43;  
  
**答案**：B  
**解析**：最短寻道时间优先磁盘调度算法即选择距离当前磁头最近的磁道进行访问：即贪心算法。此外，还有先来先服务、扫描算法、循环扫描算法等调度算法