**第一名：Union-find**  
严格地说，并查集是一种[数据结构](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%CA%FD%BE%DD%BD%E1%B9%B9&k0=%CA%FD%BE%DD%BD%E1%B9%B9&kdi0=0&luki=9&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)，它专门用来处理集合的合并操作和查询操作。并查集巧妙地借用了树结构，使得[编程](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%B1%E0%B3%CC&k0=%B1%E0%B3%CC&kdi0=0&luki=8&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)复杂度降低到了令人难以置信的地步；用上一些递归技巧后，各种操作几乎都能用两行代码搞定。而路径压缩的好主意，更是整个数据结构的画龙点睛之笔。并查集的效率极高，单次操作的时间复杂度几乎可以看作是常数级别；但由于数据结构的实际行为难以预测，精确的时间复杂度分析需要用到不少高深的技巧。  
**第二名：Knuth-Morris-Pratt字符串匹配算法**  
    关于此算法的介绍，请参考此文：六、教你从头到尾彻底理解KMP算法。KMP算法曾经落选于二十世纪最伟大的十大算法，但人们显然不能接受，如此漂亮、高效的KMP算法竟然会落选。所以，此次最终投票产出生，KMP算法排到了第二名。  
**第三名：BFPRT 算法**  
    1973 年，Blum、Floyd、Pratt、Rivest、Tarjan集体出动，合写了一篇题为 “Time bounds for selection” 的论文，给出了一种在数组中选出第 k 大元素的算法，俗称"中位数之中位数算法"。依靠一种精心设计的 pivot 选取方法，该算法从理论上保证了最坏情形下的线性时间复杂度，打败了平均线性、最坏 O(n^2) 复杂度的传统算法。一群大牛把递归算法的复杂度分析玩弄于骨掌股掌之间，构造出了一个当之无愧的来自圣经的算法。  
    我在这里简单介绍下在数组中选出第k大元素的时间复杂度为O（N）的算法：  
    类似快排中的分割算法：  
每次分割后都能返回枢纽点在数组中的位置s,然后比较s与k的大小  
若大的话，则再次递归划分array，  
小的话，就递归array //s为中间枢纽点元素。  
否则返回array，就是partition中返回的值。 //就是要找到这个s。  
找到符合要求的s值后，再遍历输出比s小的那一边的元素。  
    各位可参考在：算法导论上，第九章中，以期望线性时间做选择，一节中，  
我找到了这个 寻找数组中第k小的元素的，平均时间复杂度为O（N）的证明：上述程序的期望运行时间，最后证明可得O(n)，且假定元素是不同的。  
**第四名：Quicksort（快速排序）**  
快速排序算法几乎涵盖了所有经典算法的所有榜单。它曾获选二十世纪最伟大的十大算法（参考这：细数二十世纪最伟大的10大算法）。关于快速排序算法的具体介绍，请参考我写的这篇文章：一之续、快速排序算法的深入分析，及十二、快速排序算法之所有版本的c/[c++](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=c%2B%2B&k0=c%2B%2B&kdi0=0&luki=7&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)实现。  
**第五名：Floyd-Warshall all-pairs最短路径算法**  
关于此算法的介绍，可参考我写的此文：几个最短路径算法比较（http://blog.csdn.net/v\_JULY\_v/archive/2011/02/12/6181485.aspx）。  
d: 二维数组. d最小花费、或最短路径的邻边。  
for k from 1 to n:  
for i from 1 to n:  
for j from 1 to n:  
d = min(d, d + d)  
**第六名：Gentry's Fully Homomorphic Encryption Scheme（绅士完全同态加密机制）算法。**  
此算法很漂亮，它允许第三方执行任意加密[数据](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%CA%FD%BE%DD&k0=%CA%FD%BE%DD&kdi0=0&luki=3&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)运算得不到私钥（不是很了解）。  
**第七名：Depth First Search、Breadth First Search（深度、广度优先搜索）**  
它们是许多其他算法的基础。关于深度、广度优先搜索算法的具体介绍，请参考此文：教你通透彻底理解：BFS和DFS优先搜索算法。  
**第八名：Miller-Rabin作的类似的试验测试**  
这个想法是利用素数的性质(如使用费马大定理)的小概率寻找见证不数素数。如果没有证据是足够的随机检验后发现,这一数字为素数。  
**第九名：Binary Search （二分查找）**  
在一个有序的集合中查找元素，可以使用二分查找算法，也叫二分搜索。二分查找算法先比较位于集合中间位置的元素与键的大小，有三种情况（假设集合是从小到大排列的）：  
1.键小于中间位置的元素，则匹配元素必在左边（如果有的话），于是对左边的区域应用二分搜索。  
2.键等于中间位置的元素，所以元素找到。  
3.键大于中间位置的元素，则匹配元素必在右边（如果有的话），于是对右边的区域应用二分搜索。  
另外，当集合为空，则代表找不到。  
**第十名：Huffman coding（霍夫曼编码）**[霍夫曼](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%BB%F4%B7%F2%C2%FC&k0=%BB%F4%B7%F2%C2%FC&kdi0=0&luki=2&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)编码(Huffman Coding)是一种编码方式，是一种用于无损[数据](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%CA%FD%BE%DD&k0=%CA%FD%BE%DD&kdi0=0&luki=3&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)压缩的熵编码（权编码）算法。1952年，David A. Huffman在麻省理工攻读博士时所发明的，并发表于《一种构建极小多余编码的方法》（A Method for the Construction of Minimum-Redundancy Codes）一文。  
**十一、Cooley-Tukey FFT算法。快速**[**傅里叶变换**](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%B8%B5%C0%EF%D2%B6%B1%E4%BB%BB&k0=%B8%B5%C0%EF%D2%B6%B1%E4%BB%BB&kdi0=0&luki=5&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)**算法。**关于傅里叶变换算法的介绍，请参考此文：十、从头到尾彻底理解傅里叶变换算法、上，及十、从头到尾彻底理解傅里叶变换算法、下。  
**十二、linear programming，线性规划。**  
**十三、Dijkstra 算法。**与上第五一样，又一种最短路径算法。具体介绍，请参考：二之续、彻底理解Dijkstra算法，和二（再续）、Dijkstra 算法+fibonacci堆的逐步c实现。  
**十四、Merge Sort。**归并排序。  
**十五、Ford–Fulkerson算法。**网络最大流算法。  
**十六、辗转相除法。**  
      在数学中，辗转相除法，又称欧几里得算法，是求最大公约数的算法，即求两个正整数之最大公因子的算法。此算法作为TAOCP第一个算法被阐述，足见此算法被重视的程度。它是已知最古老的算法, 其可追溯至3000年前。辗转相除法首次出现于欧几里得的《几何原本》（第VII卷，命题i和ii）中，而在中国则可以追溯至东汉出现的《九章算术》。扩展的辗转相除法则构造性地证明了，对任意整数a和b ，存在一对x、y使得 ax + by = gcd(a, b) 。  
**十七、RSA加密演算法。**一种加密算法，日后再做详细介绍。  
**十八、遗传算法。**可参考本人写的关于GA 算法的这篇文章：七、遗传算法 透析GA本质。  
**十九、最大期望（EM）算法。**  
    此[算法](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%CB%E3%B7%A8&k0=%CB%E3%B7%A8&kdi0=0&luki=10&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)入选[数据挖掘](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%CA%FD%BE%DD%CD%DA%BE%F2&k0=%CA%FD%BE%DD%CD%DA%BE%F2&kdi0=0&luki=1&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)领域十大经典算法。在统计计算中，最大期望（EM）算法是在概率（probabilistic）[模型](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%C4%A3%D0%CD&k0=%C4%A3%D0%CD&kdi0=0&luki=4&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)中寻找参数最大似然估计的算法，其中概率模型依赖于无法观测的隐藏变量（Latent Variable）。最大期望经常用在机器学习和计算机视觉的数据聚类（Data Clustering）领域。最大期望算法经过两个步骤交替进行计算，第一步是计算期望（E），利用对隐藏变量的现有估计值，计算其最大似然估计值；第二步是最大化（M），最大化在 E 步上求得的最大似然值来计算参数的值。M 步上找到的参数估计值被用于下一个 E 步计算中，这个过程不断交替进行。  
**二十、数据压缩**  
    [数据](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%CA%FD%BE%DD&k0=%CA%FD%BE%DD&kdi0=0&luki=3&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)压缩是通过减少计算机中所存储数据或者通信传播中数据的冗余度，达到增大数据密度，最终使数据的存储空间减少的技术。数据压缩在文件存储和分布式系统领域有着十分广泛的应用。数据压缩也代表着尺寸媒介容量的增大和网络带宽的扩展。  
**二十一、Hash函数**  
    Hash，一般翻译做“散列”，也有直接音译为“哈希”的，就是把任意长度的输入（又叫做预映射， pre-image），通过散列算法，变换成固定长度的输出，该输出就是散列值。关于hash表的详细阐述，请参考此篇文章：十一、从头到尾彻底解析Hash表算法。  
**二十二、Dynamic Programming（动态规划）**。关于动态规划的粗略介绍，请参考此文：三、dynamic programming。  
**二十三、堆排序算法。**  
    堆排序算法作为一种快速稳定的算法，其平均时间复杂度（最坏也为）O（n\*lgn）。当然，在实际应用中，一个实现的好的快速排序算法仍然要优于堆排序算法。不过，堆[数据结构](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%CA%FD%BE%DD%BD%E1%B9%B9&k0=%CA%FD%BE%DD%BD%E1%B9%B9&kdi0=0&luki=9&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)还可以作为高效的优先级队列。对堆排序算法作简单了解，可参考这：堆排序算法。  
  
**二十四、递归与回溯**[**算法**](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%CB%E3%B7%A8&k0=%CB%E3%B7%A8&kdi0=0&luki=10&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)。此俩个算法，相信各位比较熟悉，在此不做赘述。    
**二十五、最长公共子序列**  
    最长公共子序列，英文缩写为LCS（Longest Common Subsequence）。其定义是，一个数列 S ，如果分别是两个或多个已知数列的子序列，且是所有符合此条件序列中最长的，则 S 称为已知序列的最长公共子序列。  
动态规划的一个计算最长公共子序列的方法如下：  
    以两个序列 X、Y 为例子：  
设有二维数组 f 表示 X 的 i 位和 Y 的 j 位之前的最长公共子序列的长度，则有：  
      f = same(1,1)  
      f*= max{f+same(i,j)，f，f}  
    其中，same(a,b)当 X 的第 a 位与 Y 的第 b 位完全相同时为“1”，否则为“0”。  
此时，f中最大的数便是 X 和 Y 的最长公共子序列的长度，依据该数组回溯，便可找出最长公共子序列。  
    该算法的空间、时间复杂度均为O(n2)，经过优化后，空间复杂度可为O(n)，时间复杂度为O(nlogn)。更多详情，参见之前写的一篇拙文（不过，鉴于写的糟，日后会重写）：三、dynamic programming。****二十六、红黑树的***[***算法***](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%CB%E3%B7%A8&k0=%CB%E3%B7%A8&kdi0=0&luki=10&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)***与实现*** *关于红黑树，linux内核中有实现，本BLOG内也已经写了4篇红黑树系列的文章。详情，请参考：五（续）、教你透彻了解红黑树。****二十七、A\*搜寻算法。*** *相对于BFS、Dijkstra 等算法，A\*搜寻算法作为一种高效的最短路径搜索算法，如今，已得到日益广泛的应用。初步了解A\*搜寻算法的高效及与其它最短路径算法的比较，请参考此文：一（续）、A\*，Dijkstra，BFS算法性能比较及A\*算法的应用。****二十八、图像特征提取与匹配之SIFT算法*** *sift，尺度不变特征转换，是一种电脑视觉的*[*算法*](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=%CB%E3%B7%A8&k0=%CB%E3%B7%A8&kdi0=0&luki=10&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)*用来侦测与描述影像中的局部性特征，它在空间尺度中寻找极值点，并提取出其位置、尺度、旋转不变量，此算法由 David Lowe 在1999年所发表，2004年完善总结。关于此算法，请参考如下，粗略介绍：九、图像特征提取与匹配之SIFT算法，利用第三方库编译过程：九（续）、sift算法的编译与实现，*[*c语言*](http://cpro.baidu.com/cpro/ui/uijs.php?adclass=0&app_id=0&c=news&cf=1001&ch=0&di=8&fv=18&is_app=0&jk=ed21f4bff96e82f9&k=c%D3%EF%D1%D4&k0=c%D3%EF%D1%D4&kdi0=0&luki=6&n=10&p=baidu&q=baidusiteerror_cpr&rb=0&rs=1&seller_id=1&sid=f9826ef9bff421ed&ssp2=1&stid=0&t=tpclicked3_hc&td=1615258&tu=u1615258&u=http%3A%2F%2Femuch.net%2Fhtml%2F201111%2F3733337.html&urlid=0)*一步一步实现sift算法：九之再续：一步一步用c语言实现sift算法、上，及九之再续：教你一步一步用c语言实现sift算法、下。*

**九章算法-面试题总结**

## *1 落单的数*

### *题目描述：*

*有2n+1个数，其中2n个数两两成对，1个数落单，找出这个数。要求O(n)的时间复杂度，O(1)的空间复杂度。*

*进阶问题：如果有2n+2个数，其中有2个数落单，该怎么办？*

### *分析*

*初阶：将2n+1个数异或起来，相同的数会抵消，异或的答案就是要找的数。*

*进阶：假设两个不同的数是a和b，并且a!=b，将2n+2个数异或起来就会得到c=a xor b，并且c不等于0。因此在c的二进制位中找到一个为1的位，可推断在这位上a和b分别为0和1，因此将2n+2个数分为该位位0的组和该位为1的组，两组中各自会包含2n’+1个数和2n’’+1个数，用初阶的算法即可解决。*

## *2 抄书问题*

### *题目描述：*

*有n本书和k个抄写员。要求n本书必须连续的分配给这k个抄写员抄写。也就是说前a1本书分给第一个抄写员，接下来a2本书分给第二个抄写员，如此类推（a1,a2需要你的算法来决定）。给定n,k和每本书的页数p1,p2..pn，假定每个抄写员速度一样（每分钟1页），k个抄写员同时开始抄写，问最少需要多少时间能够将所有书全部抄写完工？（提示：本题有很多种算法可以在不同的时间复杂度下解决，需要尽可能的想到所有的方法）*

### *分析*

*解法1：动态规划 设f[i][j]代表前i本书分给j个抄写员抄完的最少耗时。答案就是f[n][k]。状态转移方程f[i][j] = min{max(f[x][j-1], sum(x+1, i)), j<x<i}。其中x是在枚举第j个抄写员是从哪本书开始抄写。 时间复杂度O(n^2\*k)*

*解法2：动态规划+决策单调。 同上一解法，但在x的枚举上进行优化，设s[i][j]为使得f[i][j]获得最优值的x是多少。根据四边形不等式原理，有s[i][j-1]>=s[i][j]>=s[i-1][j]。因此x这一层的枚举不再是每次都是n而是总共加起来n。 时间复杂度O(n\*k)*

*解法3：二分答案。二分最慢的时间，然后尝试一本本的加进来，加满了就给一个抄写员。看最后需要的抄写员数目是多余k个还是少于k个，然后来决定是将答案往上调整还是往下调整。 时间复杂度O( n log SUM(pi) )*

## *3 找坏球*

### *题目描述：*

*有12个球，1个没有砝码的天秤。其中有11个球的重量是一样的，另外1个是坏球，和其他球的重量不一样，但无法确定是轻了还是重了。请问如何用天秤称3次，就找到坏球并确定是轻了还是重了。（没有砝码的天秤只能比较出两边谁重谁轻或是重量相等，无法求得具体的重量差）*

### *分析*

*12个球，编号A=（1，2，3，4 ）B=（5，6，7，8）C=（9，10，11，12）*

*分为三组:A, B, C。*

* *先比较A,B,如果A与B平衡，则A，B中均为好球*
  + *比较5，6，7与9，10，11*
    - *若平衡，则坏球为8或12*
      * *比较8与任何一个好球，平衡，坏球为12；不平，坏球为8。*
    - *若不平，则目前可以知道是坏的球比好球是重还是轻，假设为重*
      * *比较9，10，若平衡，则坏球为12；不平，坏球为重的那个*
* *若A，B不平，则C为好球*
  + *比较1，2，5 与 3，4，6（交叉，这玩意面试的时候能想到？）*
    - *若平衡，则坏球在7，8之间，且之前已经得知轻重的一个关系，再比较一次7，8*
    - *若不平衡，或者1，2为坏球，或者5，6为坏球，再比较一次。*

## *4 索引比例*

### *题目描述：*

*估算Baidu和Google的网页索引数量之比*

### *分析*

*我们可以假设能够通过搜索引擎做到如下的两件事：*

1. *随机取到一个网页*
2. *判断某个网页(url)是否被索引*

*因此，在Baidu上多次随机关键词进行搜索，获取到每个关键词对应结果的若干网页信息（url），将这些url在Google上查找是否被索引到。从而得到Baidu网页中Google索引的的比例为1/B。*

*对Google做同样的事情，得到Google网页中被Baidu索引的比例1/G。由此可知Baidu和Google的索引比例为B:G*

## *5 第k大的数*

### *题目描述*

*初阶：有两个数组A和B，假设A和B已经有序（从大到小），求A和B数组中所有数的第K大。*

*进阶：有N台机器，每台机器上有一个有序大数组，需要求得所有机器上所有数中的第K大。注意，需要考虑N台机器的并行计算能力。*

### *分析*

*初阶：比较A[k/2]和B[k/2]，如果A[k/2]>=B[k/2]那么A的前k/2个数一定都在前k-1大中，将A数组前k/2个数扔掉，反之扔掉B的前k/2个数。将k减小k/2。重复上述操作直到k=1。比较A和B的第一个数即可得到结果。时间复杂度O(logk) Leetcode原题，据说极其高频*

*进阶：二分答案S，将S广播给每台机器，每台机器用二分法求得有多少比该数小的数。汇总结果后可判断是该将S往上还是往下调整。*

*面试官角度：*

*初阶问题是一个比较难度大的算法题。需要有一定的算法训练功底。主要用到的思想是递归。首先容易想到的方法是合并两个数组（见面试题5，有序数组的合并），这样复杂度为O(k)，那么答出这个以后，面试官会问你，还有更好的方法么？这个时候就要往O(logk)的思路去想，O(logk)就意味着需要用一种方法每次将k的规模减小一半，于是想到，每次要扔掉一个数组或两个数组中的k/2个数，于是想到去比较A[k/2]和B[k/2]，仔细思考比较结果，然后想到较大的那一方的前k/2个数一定都在前k-1大的数中，于是可以扔掉。*

*进阶问题的考察点是逆向思维。二分答案是一种常见的算法思路（见面试题2 抄书问题），所以当你想不出题目的时候，往往可以试试看是否可以二分答案。因为需要发挥N台机器的并行计算能力，所以想到让每台机器互不相关的做一件事情，然后将结果汇总来判断。*

### *进阶题伪代码*

*int FindK(vector<vector<int> > & mq, int N, int k)*

*{*

*int ans\_upper = INT\_MAX;*

*int ans\_lower = INT\_MIN;*

*int sum\_len = 0;*

*while(sum\_len != k)*

*{*

*ans = ans\_lower + (ans\_upper - ans\_lower) /2;*

*for (int i=0; i<N; i++)*

*{*

*sum\_len += left\_count\_binarySearch(mq[i], ans);*

*}*

*if (sum\_len > k) ans\_upper = ans;*

*if (sum\_len < k) ans\_lower = ans;*

*}*

*return ans;*

*}*

## *6 前k大的和，这题很有意思*

### *题目描述*

*初阶：有两个数组A和B，每个数组有k个数，从两个数组中各取一个数加起来可以组成k\*k个和，求这些和中的前k大。*

*进阶：有N个数组，每个数组有k个数，从N个数组中各取一个数加起来可以组成k^n个和，求这些和中的前k大。*

### *分析*

| *~* | *9* | *7* |
| --- | --- | --- |
| *11* | *20* | *18* |
| *7* | *16* | *14* |
| *1* | *10* | *8* |
| *0* | *9* | *7* |

*可以这个问题转化为N个有序数组的合并问题，每个数组为：*

*因此，维护一个包含N个元素的最大堆，然后每取一个元素T，T所在列的下一个元素入堆，这样循环取K个数，即完成了求TopK的问题。*

#### *进阶：*

*先求1，2前K大，然后再与下一个求前K大。*

## *7 赛马问题*

### *题目描述*

*有25匹马，有一个5个赛道的马场，每场比赛可以决出5匹马的排名，假设每匹马发挥稳定，且不会出现名次相同的情况。问，如果要知道25匹马中跑得最快的马，需要几场比赛？如果需要知道跑得第二快的马，需要几场比赛？第三快的呢？*

### *分析*

1. *最快的，需要6次。*
   * *每五匹赛一次（5次），每次的第一名，再一起赛一次（1次）*
2. *第二快的，需要7次。*
   * *每五匹赛一次（5次），每次的第一名，再一起赛一次（1次）*
   * *最快的那组的第二名，与上次的第二名，跑一次。(1次)*
3. *第三快的，需要7次。*
   * *每五匹赛一次（5次），每次的第一名，再一起赛一次（1次）*
   * *最快的那组的第二、三名，与上次的第二名那组里的第二名，与上次的第二、第三名一起跑一次。*

## *8 最大子区间/矩阵*

### *题目描述*

*初阶：数组A中有N个数，需要找到A的最大子区间，使得区间内的数和最大。即找到0<=i<=j<N，使得A[i]+A[i+1] … + A[j]最大。A中元素有正有负。*

*进阶：矩阵A中有N\*N个数，需要找到A的最大的子矩阵。*

### *分析*

*第一个是经典的连续子序列问题，DP或者转化为前缀和数组进行贪心。*

*第二个，枚举上下行，中间压缩为一个值，再采用第一个问题中的方法求解，复杂度O(n^3)*

## *9 从输入流中随机取记录*

### *题目描述*

*有一个很大很大的输入流，大到没有存储器可以将其存储下来，而且只输入一次，如何从这个输入流中等概率随机取得m个记录。*

### *分析*

*维护一个内存空间，存放前m个记录，然后遇到第k元素，从m个记录中，随机抽取一个元素，然后以m/k的概率替换这个元素。*

#### *简单证明*

*假设数据流一共N个元素，设第k个元素最后存在于所选取的记录里，则当遇到第k个元素时，k一定被替换，而k+1到最后N一定不被替换。*

*第k个元素不管替换了之前的哪一个元素，那肯定会留下来，因此概率为m/k。而后面第j个元素会替换k的概率是，j要替换，且随机从m个元素中选到了k。其概率为*

*因此，第k个元素，最终仍会存在的概率为：*

*因此每个元素被取得的概率相等。*

## *10 最常访问IP*

### *题目描述*

*给你一个海量的日志数据，提取出某日访问网站次数最多的IP地址。*

### *分析*

*将日志文件划分成适度大小的M份存放到处理节点。*

*每个map节点所完成的工作：统计访问百度的ip的出现频度（比较像统计词频，使用字典树），并完成相同ip的合并(combine)。*

*map节点将其计算的中间结果partition到R个区域，并告知master存储位置，所有map节点工作完成之后，reduce节点首先读入数据，然后以中间结果的key排序，对于相同key的中间结果调用用户的reduce函数，输出。*

*扫描reduce节点输出的R个文件一遍，可获得访问网站度次数最多的ip。*

*面试官角度：*

*该问题是经典的Map-Reduce问题。一般除了问最常访问，还可能会问最常访问的K个IP。一般来说，遇到这个问题要先回答Map-Reduce的解法。因为这是最常见的解法也是一般面试官的考点。如果面试官水平高一点，要进一步问你有没有其他解法的话，该问题有很多概率算法。能够在极少的空间复杂度内，扫描一遍log即可得到Top k Frequent Items（在一定的概率内）。有兴趣的读者，可以搜搜“Sticky Sampling”，”Lossy Counting”这两个算法。*

[*九章算法-面试题总结(算法、强化算法、系统设计高清视频观看)*](http://www.bubuko.com/infodetail-1913585.html)

*标签：*[*包含*](http://www.bubuko.com/so/1/%E5%8C%85%E5%90%AB)[*贪心*](http://www.bubuko.com/so/1/%E8%B4%AA%E5%BF%83)[*抽取*](http://www.bubuko.com/so/1/%E6%8A%BD%E5%8F%96)[*tin*](http://www.bubuko.com/so/1/tin)[*内存*](http://www.bubuko.com/so/1/%E5%86%85%E5%AD%98)[*代码*](http://www.bubuko.com/so/1/%E4%BB%A3%E7%A0%81)[*海量*](http://www.bubuko.com/so/1/%E6%B5%B7%E9%87%8F)[*发挥*](http://www.bubuko.com/so/1/%E5%8F%91%E6%8C%A5)[*有序*](http://www.bubuko.com/so/1/%E6%9C%89%E5%BA%8F)

## *1 落单的数*

### *题目描述：*

*有2n+1个数，其中2n个数两两成对，1个数落单，找出这个数。要求O(n)的时间复杂度，O(1)的空间复杂度。*

*进阶问题：如果有2n+2个数，其中有2个数落单，该怎么办？*

### *分析*

*初阶：将2n+1个数异或起来，相同的数会抵消，异或的答案就是要找的数。*

*进阶：假设两个不同的数是a和b，并且a!=b，将2n+2个数异或起来就会得到c=a xor b，并且c不等于0。因此在c的二进制位中找到一个为1的位，可推断在这位上a和b分别为0和1，因此将2n+2个数分为该位位0的组和该位为1的组，两组中各自会包含2n’+1个数和2n’’+1个数，用初阶的算法即可解决。*

## *2 抄书问题*

### *题目描述：*

*有n本书和k个抄写员。要求n本书必须连续的分配给这k个抄写员抄写。也就是说前a1本书分给第一个抄写员，接下来a2本书分给第二个抄写员，如此类推（a1,a2需要你的算法来决定）。给定n,k和每本书的页数p1,p2..pn，假定每个抄写员速度一样（每分钟1页），k个抄写员同时开始抄写，问最少需要多少时间能够将所有书全部抄写完工？（提示：本题有很多种算法可以在不同的时间复杂度下解决，需要尽可能的想到所有的方法）*

### *分析*

*解法1：动态规划 设f[i][j]代表前i本书分给j个抄写员抄完的最少耗时。答案就是f[n][k]。状态转移方程f[i][j] = min{max(f[x][j-1], sum(x+1, i)), j<x<i}。其中x是在枚举第j个抄写员是从哪本书开始抄写。 时间复杂度O(n^2\*k)*

*解法2：动态规划+决策单调。 同上一解法，但在x的枚举上进行优化，设s[i][j]为使得f[i][j]获得最优值的x是多少。根据四边形不等式原理，有s[i][j-1]>=s[i][j]>=s[i-1][j]。因此x这一层的枚举不再是每次都是n而是总共加起来n。 时间复杂度O(n\*k)*

*解法3：二分答案。二分最慢的时间，然后尝试一本本的加进来，加满了就给一个抄写员。看最后需要的抄写员数目是多余k个还是少于k个，然后来决定是将答案往上调整还是往下调整。 时间复杂度O( n log SUM(pi) )*

## *3 找坏球*

### *题目描述：*

*有12个球，1个没有砝码的天秤。其中有11个球的重量是一样的，另外1个是坏球，和其他球的重量不一样，但无法确定是轻了还是重了。请问如何用天秤称3次，就找到坏球并确定是轻了还是重了。（没有砝码的天秤只能比较出两边谁重谁轻或是重量相等，无法求得具体的重量差）*

### *分析*

*12个球，编号A=（1，2，3，4 ）B=（5，6，7，8）C=（9，10，11，12）*

*分为三组:A, B, C。*

* *先比较A,B,如果A与B平衡，则A，B中均为好球*
  + *比较5，6，7与9，10，11*
    - *若平衡，则坏球为8或12*
      * *比较8与任何一个好球，平衡，坏球为12；不平，坏球为8。*
    - *若不平，则目前可以知道是坏的球比好球是重还是轻，假设为重*
      * *比较9，10，若平衡，则坏球为12；不平，坏球为重的那个*
* *若A，B不平，则C为好球*
  + *比较1，2，5 与 3，4，6（交叉，这玩意面试的时候能想到？）*
    - *若平衡，则坏球在7，8之间，且之前已经得知轻重的一个关系，再比较一次7，8*
    - *若不平衡，或者1，2为坏球，或者5，6为坏球，再比较一次。*

## *4 索引比例*

### *题目描述：*

*估算Baidu和Google的网页索引数量之比*

### *分析*

*我们可以假设能够通过搜索引擎做到如下的两件事：*

1. *随机取到一个网页*
2. *判断某个网页(url)是否被索引*

*因此，在Baidu上多次随机关键词进行搜索，获取到每个关键词对应结果的若干网页信息（url），将这些url在Google上查找是否被索引到。从而得到Baidu网页中Google索引的的比例为1/B。*

*对Google做同样的事情，得到Google网页中被Baidu索引的比例1/G。由此可知Baidu和Google的索引比例为B:G*

## *5 第k大的数*

### *题目描述*

*初阶：有两个数组A和B，假设A和B已经有序（从大到小），求A和B数组中所有数的第K大。*

*进阶：有N台机器，每台机器上有一个有序大数组，需要求得所有机器上所有数中的第K大。注意，需要考虑N台机器的并行计算能力。*

### *分析*

*初阶：比较A[k/2]和B[k/2]，如果A[k/2]>=B[k/2]那么A的前k/2个数一定都在前k-1大中，将A数组前k/2个数扔掉，反之扔掉B的前k/2个数。将k减小k/2。重复上述操作直到k=1。比较A和B的第一个数即可得到结果。时间复杂度O(logk) Leetcode原题，据说极其高频*

*进阶：二分答案S，将S广播给每台机器，每台机器用二分法求得有多少比该数小的数。汇总结果后可判断是该将S往上还是往下调整。*

*面试官角度：*

*初阶问题是一个比较难度大的算法题。需要有一定的算法训练功底。主要用到的思想是递归。首先容易想到的方法是合并两个数组（见面试题5，有序数组的合并），这样复杂度为O(k)，那么答出这个以后，面试官会问你，还有更好的方法么？这个时候就要往O(logk)的思路去想，O(logk)就意味着需要用一种方法每次将k的规模减小一半，于是想到，每次要扔掉一个数组或两个数组中的k/2个数，于是想到去比较A[k/2]和B[k/2]，仔细思考比较结果，然后想到较大的那一方的前k/2个数一定都在前k-1大的数中，于是可以扔掉。*

*进阶问题的考察点是逆向思维。二分答案是一种常见的算法思路（见面试题2 抄书问题），所以当你想不出题目的时候，往往可以试试看是否可以二分答案。因为需要发挥N台机器的并行计算能力，所以想到让每台机器互不相关的做一件事情，然后将结果汇总来判断。*

### *进阶题伪代码*

*int FindK(vector<vector<int> > & mq, int N, int k)*

*{*

*int ans\_upper = INT\_MAX;*

*int ans\_lower = INT\_MIN;*

*int sum\_len = 0;*

*while(sum\_len != k)*

*{*

*ans = ans\_lower + (ans\_upper - ans\_lower) /2;*

*for (int i=0; i<N; i++)*

*{*

*sum\_len += left\_count\_binarySearch(mq[i], ans);*

*}*

*if (sum\_len > k) ans\_upper = ans;*

*if (sum\_len < k) ans\_lower = ans;*

*}*

*return ans;*

*}*

## *6 前k大的和，这题很有意思*

### *题目描述*

*初阶：有两个数组A和B，每个数组有k个数，从两个数组中各取一个数加起来可以组成k\*k个和，求这些和中的前k大。*

*进阶：有N个数组，每个数组有k个数，从N个数组中各取一个数加起来可以组成k^n个和，求这些和中的前k大。*

### *分析*

| *~* | *9* | *7* |
| --- | --- | --- |
| *11* | *20* | *18* |
| *7* | *16* | *14* |
| *1* | *10* | *8* |
| *0* | *9* | *7* |

*可以这个问题转化为N个有序数组的合并问题，每个数组为：*

*因此，维护一个包含N个元素的最大堆，然后每取一个元素T，T所在列的下一个元素入堆，这样循环取K个数，即完成了求TopK的问题。*

#### *进阶：*

*先求1，2前K大，然后再与下一个求前K大。*

## *7 赛马问题*

### *题目描述*

*有25匹马，有一个5个赛道的马场，每场比赛可以决出5匹马的排名，假设每匹马发挥稳定，且不会出现名次相同的情况。问，如果要知道25匹马中跑得最快的马，需要几场比赛？如果需要知道跑得第二快的马，需要几场比赛？第三快的呢？*

### *分析*

1. *最快的，需要6次。*
   * *每五匹赛一次（5次），每次的第一名，再一起赛一次（1次）*
2. *第二快的，需要7次。*
   * *每五匹赛一次（5次），每次的第一名，再一起赛一次（1次）*
   * *最快的那组的第二名，与上次的第二名，跑一次。(1次)*
3. *第三快的，需要7次。*
   * *每五匹赛一次（5次），每次的第一名，再一起赛一次（1次）*
   * *最快的那组的第二、三名，与上次的第二名那组里的第二名，与上次的第二、第三名一起跑一次。*

## *8 最大子区间/矩阵*

### *题目描述*

*初阶：数组A中有N个数，需要找到A的最大子区间，使得区间内的数和最大。即找到0<=i<=j<N，使得A[i]+A[i+1] … + A[j]最大。A中元素有正有负。*

*进阶：矩阵A中有N\*N个数，需要找到A的最大的子矩阵。*

### *分析*

*第一个是经典的连续子序列问题，DP或者转化为前缀和数组进行贪心。*

*第二个，枚举上下行，中间压缩为一个值，再采用第一个问题中的方法求解，复杂度O(n^3)*

## *9 从输入流中随机取记录*

### *题目描述*

*有一个很大很大的输入流，大到没有存储器可以将其存储下来，而且只输入一次，如何从这个输入流中等概率随机取得m个记录。*

### *分析*

*维护一个内存空间，存放前m个记录，然后遇到第k元素，从m个记录中，随机抽取一个元素，然后以m/k的概率替换这个元素。*

#### *简单证明*

*假设数据流一共N个元素，设第k个元素最后存在于所选取的记录里，则当遇到第k个元素时，k一定被替换，而k+1到最后N一定不被替换。*

*第k个元素不管替换了之前的哪一个元素，那肯定会留下来，因此概率为m/k。而后面第j个元素会替换k的概率是，j要替换，且随机从m个元素中选到了k。其概率为*

*因此，第k个元素，最终仍会存在的概率为：*

*因此每个元素被取得的概率相等。*

## *10 最常访问IP*

### *题目描述*

*给你一个海量的日志数据，提取出某日访问网站次数最多的IP地址。*

### *分析*

*将日志文件划分成适度大小的M份存放到处理节点。*

*每个map节点所完成的工作：统计访问百度的ip的出现频度（比较像统计词频，使用字典树），并完成相同ip的合并(combine)。*

*map节点将其计算的中间结果partition到R个区域，并告知master存储位置，所有map节点工作完成之后，reduce节点首先读入数据，然后以中间结果的key排序，对于相同key的中间结果调用用户的reduce函数，输出。*

*扫描reduce节点输出的R个文件一遍，可获得访问网站度次数最多的ip。*

*面试官角度：*

*该问题是经典的Map-Reduce问题。一般除了问最常访问，还可能会问最常访问的K个IP。一般来说，遇到这个问题要先回答Map-Reduce的解法。因为这是最常见的解法也是一般面试官的考点。如果面试官水平高一点，要进一步问你有没有其他解法的话，该问题有很多概率算法。能够在极少的空间复杂度内，扫描一遍log即可得到Top k Frequent Items（在一定的概率内）。有兴趣的读者，可以搜搜“Sticky Sampling”，”Lossy Counting”这两个算法。*