发信人: phylips (星星||一年磨十剑), 信区: Algorithm
标　题: 面试题目-大数据量专题
发信站: 兵马俑 BBS (Thu Nov 26 16:30:44 2009), 本站 (bbs.xjtu.edu.cn)

1. 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。

2. 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

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 个字节，内存限制大小是 1M。返回频数最高的 100 个词

4.海量日志数据，提取出某日访问百度次数最多的那个IP。

5.2.5 亿个整数中找出不重复的整数，内存空间不足以容纳这 2.5 亿个整数。

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。

7.怎么在海量数据中找出重复次数最多的一个

8.上千万or亿数据（有重复），统计其中出现次数最多的前N个数据。

统计可以用hash,二叉数,trie树。对统计结果用堆求出现的前n大数据。增加点限制可以提高效率，比如 出现次数>数据总数 / N的一定是在前N个之内

9.1000 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？

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十个词。请给出思想，给时间复杂度分析。

11.一个文本文件，也是找出前十个最经常出现的词，但这次文件比较长，说是上亿行或者十亿行，总之无法一次读入内存，问最优解。

12.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都可能重复要按照query的频度排序

13.100w个数中找最大的前 100 个数

14.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。
（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。

15.一共有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)？

本文由phylips@bmy收集整理，转载请注明出处 http://bbs.xjtu.edu.cn
谢谢合作。
--
如果可以
愿把这生命燃烧
只留下星星的传说
悲伤而让人怀念
==
※ 来源:.兵马俑BBS http://bbs.xjtu.edu.cn [FROM: 219.224.191.247]

发信人: phylips (星星||一年磨十剑), 信区: Algorithm
标　题: 大数据量，海量数据 处理方法总结
发信站: 兵马俑BBS (Thu Nov 26 16:32:38 2009), 本站 (bbs.xjtu.edu.cn)

最近有点忙，稍微空闲下来，发篇总结贴。

大数据量的问题是很多面试笔试中经常出现的问题，比如baidu google 腾讯这样的一些涉及到海量数据的公司经常会问到。

下面的方法是我对海量数据的处理方法进行了一个一般性的总结，当然这些方法可能并不能完全覆盖所有的问题，但是这样的一些方法也基本可以处理绝大多数遇到的问题。下面的一些问题基本直接来源于公司的面试笔试题目，方法不一定最优，如果你有更好的处理方法，欢迎与我讨论。

1.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

基本原理及要点：
对于原理来说很简单，位数组+k个独立hash函数。将hash函数对应的值的位数组置 1，查找时如果发现所有hash函数对应位都是 1 说明存在，很明显这个过程并不保证查找的结果是 100%正确的。同时也不支持删除一个已经插入的关键字，因为该关键字对应的位会牵动到其他的关键字。所以一个简单的改进就是 counting Bloom filter，用一个counter数组代替位数组，就可以支持删除了。

还有一个比较重要的问题，如何根据输入元素个数n，确定位数组m的大小及hash函数个数。当hash函数个数k=(ln2)*(m/n)时错误率最小。在错误率不大于E的情况下，m至少要等于n*lg(1/E)才能表示任意n个元素的集合。但m还应该更大些，因为还要保证bit数组里至少一半为 0，则m应该>=nlg(1/E)*lge 大概就是nlg(1/E)1.44 倍(lg表示以 2 为底的对数)。

举个例子我们假设错误率为 0.01，则此时m应大概是n的 13 倍。这样k大概是 8 个。

注意这里m与n的单位不同，m是bit为单位，而n则是以元素个数为单位(准确的说是不同元素的个数)。通常单个元素的长度都是有很多bit的。所以使用bloom filter内存上通常都是节省的。

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？

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urlip是一一对应的，就可以转换成ip，则大大简单了。

2.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。

扩展：
d-left hashing中的d是多个的意思，我们先简化这个问题，看一看 2-left hashing。2-left hashing指的是将一个哈希表分成长度相等的两半，分别叫做 T1 和T2，给T1 和T2 分别配备一个哈希函数，h1 和h2。在存储一个新的key 时，同时用两个哈希函数进行计算，得出两个地址h1[key]和h2[key]。这时需要检查T1 中的h1[key]位置和T2 中的h2[key]位置，哪一个位置已经存储的（有碰撞的）key比较多，然后将新key存储在负载少的位置。如果两边一样多，比如两个位置都为空或者都存储了一个key，就把新key 存储在左边的T1 子表中，2-left也由此而来。在查找一个key时，必须进行两次hash，同时查找两个位置。

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。

3.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

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。

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。

4.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

基本原理及要点：最大堆求前n小，最小堆求前n大。方法，比如求前n小，我们比较当前元素与最大堆里的最大元素，如果它小于最大元素，则应该替换那个最大元素。这样最后得到的n个元素就是最小的n个。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：
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。

5.双层桶划分

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。可以通过多次缩小，双层只是一个例子。

扩展：

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。

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。

6.数据库索引

适用范围：大数据量的增删改查

基本原理及要点：利用数据库的设计实现方法，对海量数据的增删改查进行处理。
扩展：
问题实例：

7.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。

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。

8.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

扩展：

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。

9.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

基本原理及要点：实现方式，节点孩子的表示方式

扩展：压缩实现。

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 。

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。

10.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

基本原理及要点：将数据交给不同的机器去处理，数据划分，结果归约。

扩展：

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:

```
 void map(String name, String document):
   // name: document name
   // document: document contents
   for each word w in document:
     EmitIntermediate(w, 1);

 void reduce(String word, Iterator partialCounts):
   // key: a word
   // values: a list of aggregated partial counts
   int result = 0;
   for each v in partialCounts:
     result += ParseInt(v);
   Emit(result);
```

Here, each document is split in words, and each word is counted initially with a "1" value by

the Map function, using the word as the result key. The framework puts together all the pairs

with the same key and feeds them to the same call to Reduce, thus this function just needs to

sum all of its input values to find the total appearances of that word.

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)？

经典问题分析

上千万or亿数据（有重复），统计其中出现次数最多的前N个数据,分两种情况：可一次读入内存，不可一次读入。

可用思路：trie树+堆，数据库索引，划分子集分别统计，hash，分布式计算，近似统计，外排序

所谓的是否能一次读入内存，实际上应该指去除重复后的数据量。如果去重后数据可以放入内存，我们可以为数据建立字典，比如通过 map，hashmap，trie，然后直接进行统计即可。当然在更新每条数据的出现次数的时候，我们可以利用一个堆来维护出现次数最多的前N个数据，当然这样导致维护次数增加，不如完全统计后在求前N大效率高。

如果数据无法放入内存。一方面我们可以考虑上面的字典方法能否被改进以适应这种情形，可以做的改变就是将字典存放到硬盘上，而不是内存，这可以参考数据库的存储方法。

当然还有更好的方法，就是可以采用分布式计算，基本上就是map-reduce过程，首先可以根据数据值或者把数据hash(md5)后的值，将数据按照范围划分到不同的机子，最好可以让数据划分后可以一次读入内存，这样不同的机子负责处理各种的数值范围，实际上就是map。得到结果后，各个机子只需拿出各自的出现次数最多的前N个数据，然后汇总，选出所有的数据中出现次数最多的前N个数据，这实际上就是reduce过程。

实际上可能想直接将数据均分到不同的机子上进行处理，这样是无法得到正确的解的。因为一个数据可能被均分到不同的机子上，而另一个则可能完全聚集到一个机子上，同时还可能存在具有相同数目的数据。比如我们要找出现次数最多的前 100 个，我们将 1000 万的数据分布到 10 台机器上，找到每台出现次数最多的前 100 个，归并之后这样不能保证找到真正的第 100 个，因为比如出现次数最多的第 100 个可能有 1 万个，但是它被分到了 10 台机子，这样在每台上只有 1 千个，假设这些机子排名在 1000 个之前的那些都是单独分布在一台机子上的，比如有 1001 个，这样本来具有 1 万个的这个就会被淘汰，即使我们让每台机子选出出现次数最多的 1000 个再归并，仍然会出错，因为可能存在大量个数为 1001 个的发生聚集。因此不能将数据随便均分到不同机子上，而是要根据hash 后的值将它们映射到不同的机子上处理，让不同的机器处理一个数值范围。

而外排序的方法会消耗大量的IO，效率不会很高。而上面的分布式方法，也可以用于单机版本，也就是将总的数据根据值的范围，划分成多个不同的子文件，

然后逐个处理。处理完毕之后再对这些单词的及其出现频率进行一个归并。实际上就可以利用一个外排序的归并过程。

另外还可以考虑近似计算，也就是我们可以通过结合自然语言属性，只将那些真正实际中出现最多的那些词作为一个字典，使得这个规模可以放入内存。

转载请注明出处：http://bbs.xjtu.edu.cn
作者phylips@bmy

参考文献：
http://blog.csdn.net/jiaomeng/archive/2007/03/08/1523940.aspx
d-Left Hashing
http://blog.csdn.net/jiaomeng/archive/2007/01/27/1495500.aspx
http://en.wikipedia.org/wiki/Bloom_filter
http://hi.baidu.com/xdzhang_china/blog/item/2847777e83fb020229388a15.html 应用Bloom Filter的几个小技巧
http://zh.wikipedia.org/wiki/%E5%80%92%E6%8E%92%E7%B4%A2%E5%BC%95

--
如果可以
愿把这生命燃烧
只留下星星的传说
悲伤而让人怀念
==

发信人: phylips (星星||一年磨十剑), 信区: Algorithm
标　题: Re: 大数据量，海量数据 处理方法总结
发信站: 兵马俑 BBS (Fri Oct　1 21:30:30 2010), 本站(bbs.xjtu.edu.cn)

第 10 条，适用范围需要更正一下，实际来说并不需要数据可以一次放入内存。

另外对于求出现次数最多的前 n 个：
“基本上就是 map-reduce 过程，首先可以根据数据值或者把数据 hash(md5)后的值，将数据按照范围划分到不同的机子，最好可以让数据划分后可以一次读入内存，这样不同的机子负责处理各种的数值范围，实际上就是 map。得到结果后，各个机子只需拿出各自的出现次数最多的前 N 个数据，然后汇总，选出所有的数据中出现次数最多的前 N 个数据，这实际上就是 reduce 过程。
”
此段描述实际上是 map reduce 模型的一个小的实现，在当利用实际的 MapReduce 实现进行计算时，实际上只是一个 wordcount。划分归并排序都是由系统处理掉的。
【 在 phylips 的大作中提到: 】
: 最近有点忙，稍微空闲下来，发篇总结贴。

发信人: appsony (懒羊羊), 信区: Algorithm
标　题: 跟风发点补充的--大数据处理
发信站: 兵马俑BBS (Fri Nov 27 19:13:23 2009), 本站(bbs.xjtu.edu.cn)

用来应付面试题，祝大家找工作顺利。

1.完美哈希函数(Perfect Hash Function)
　所谓完美哈希函数，就是指没有冲突的哈希函数，设定义域为X，值域为Y, n=|X|,m=|Y|，那么肯定有m>=n,如果对于不同的key1,key2 属于X,有h(key1)!=h(key2)，那么称h为完美哈希函数，当m=n时，h称为最小完美哈希函数（这个时候就是一一映射了）。
　相信大家在处理大规模字符串数据的时候，经常遇到这样的需求：首先需要把数据中出现的每个不同字符串分配一个唯一的整数ID，以后就用这个整数来代替这个字符串了。这个时候只要找到一个字符串的完美哈希函数，就可以解决了。
　算法：假设有 2 个随机的哈希函数h1 和h2,都将字符串映射到 0..m-1 域内，假设现在有一个g函数，使得(g(h1(str_i))+g(h2(str_i))) % n = i,那么这个哈希函数就可以作为最小完美哈希函数了。由于h1 和h2 已知，现在的目标就是要找到g函数，建立一个m个顶点的图，然后添加n条边，第i条边为(h1(str_i),h2(str_i)),边权为i，可以证明：只要这个图是一个无环图，就一定存在满足条件的g函数：每次找一个没有分配g值的顶点v，令g[v]=0，然后从这个顶点开始深度优先遍历，给其它每个点分配相应的g值。
　问题：算法最关键的问题是m值的选取，这个涉及到 2 方面的取舍：1.m值不能太大，否则g函数定义域太大，内存存不下 2.m值不能太小，否则生成的图有环的概率会非常大。
　解决方法：设置更多的随机函数，比如h1,h2,h3,这个时候哈希函数就是(g(h1(str_i))+g(h2(str_i)+g(h3(str_i))) % n = i，可以证明，此时m值不需要很大，就能使生成的图无环的概率很大。

2.排序二叉树
　排序二叉树是一个动态的数据结构，一般说的排序二叉树的用途就是动态的快速查找某一个数。但是如果我们在二叉树的结点上增加更多的信息，就能发挥更nb的作用了。
　实例：有一个在线论坛，发帖量和回复量都非常大，帖子按照新发表或者新回复的时间来排序，要你设计一个算法，来快速的选出第n页的所有帖子（假设每页显示 20 个帖子）。
　解决方法：这个就是要动态的查询一堆数里面第x大到第y大之间的所有数了，可以增加，删除，修改那些数。如果在普通的二叉排序树的结点上增加一个域，表示它的左子树中的结点数，那么就可以很好的解决这个问题了。

```
void search(tree t, int x, int y) {
  if(x > y) return;
  if(y <= t.left_num) {
      search(t.left_child, x, y);
      return;
```

```
    }
    if(x > t.left_num + 1) {
        search(t.right_child, x - t.left_num - 1, y - t.left_num - 1);
return;
    }
    search(t.left_child, x, t.left_num);
    选取t;
    search(t.right_child, 1, y - t.left_num - 1);
  }
```

注意：上面说的二叉树在面对大规模数据时，是指的平衡二叉树。

扩展：陈启峰的SBT树，有兴趣的可以去研究下。

3.树状数组

假设有n个元素的数组a,每次可以在某个元素上执行加上一个数或者减去一个数的操作，然后需要能够快速的求出a[0]+a[1]+a[2]+...a[i]。这个可以用树状数组解决，每次更新或者询问的时间复杂度都是O(logn).

应用实例：qq拼音输入法引入了等级制度，用户会不定期的发送一个积分w到服务器，然后服务器把这个w累加到用户的总积分，并快速的返回这个用户的总积分在全球的排名。

解决方法：1. 由于这个问题实质还是动态的求某个数的排名（也就是求集合中比这个数小的数有多少个），可以利用上面平衡二叉树或者SBT树来解决，但是由于用户众多，树太大，只能保存在磁盘上。

2.注意到这个问题有个显著的特点：用户量很大，但是用户的积分值不可能很大。假设用户的积分值最大为 10^6，那么开一个 10^6 的数组a,a[i]表示积分为i的用户有多少个，那么当需要给某个用户增加积分时，假设这个用户原始积分为o,那么首先使a[o]=a[o]-1,然后a[o+w]=a[o+w]+1,询问积分为i的用户的排名，实质就是求a[0]+a[1]+a[2]+...a[w-1]，这个用上面说到的树状数组就可以了。时间空间效率都非常好。

4.索引

数据库里面的聚族索引和非聚族索引

这方面的问题挺重要的，但是了解的人不是很多。一般来说，聚族索引就是数据的存放顺序和聚族索引的顺序是一致的（有时数据会直接存放到聚族索引那个磁盘页中），而非聚族索引则不然，在非聚族索引中，需要存放一个磁盘地址指向真实的数据块，而且连续的非聚族索引会对应着不连续的数据块。由于数据只可能有一种存放顺序，所以一个数据表中只能有一个聚族索引，但是可以有多个非聚族索引。

查询效率区别：由于上面说到的区别，这 2 个索引在应对不同类别的查询时，效率是不同的。一般来说，聚族索引可以很高效的应对各种查询，但是非聚族索引基本只能高效的应对结果集是少量的查询，比如select * from A where id=1。对于范围类查询，比如select * from A where id>1 and id<1000,如果id字段是非聚族索引，那么效率远远没有聚族索引高，因为数据库每找到一个索引页后，还需要单独的一次io去取数据块，而且由于非聚族索引的特点，这些数据块是不连续的，导致磁头会不停的寻道，浪费很多io时间。

--
many people think they are full of niubility, and like to play
zhuangbility, which only reflect their shability.

※ 来源:. 兵马俑BBS http://bbs.xjtu.edu.cn [FROM: 202.117.3.13]

发信人: phylips (星星||一年磨十剑), 信区: Algorithm
标　题: 面试题目 链表专题
发信站: 兵马俑BBS (Thu Apr 29 10:58:17 2010), 本站(bbs.xjtu.edu.cn)

以前在blog上写的总结，或许对笔试面试的同学会有帮助，继续发下，里面的实现仅作参考，可能还有问题，欢迎指出，谢谢。

面试的时候，书写程序要注意以下几点

1.确认了解题意，如果对题意了解不清，应该向面试人员问清楚
2.明确题意后，首先思考找到一个复杂度可以接受的正确算法，并表述出来，注意可以在草稿纸上写写划划，进行验证
3.观察复杂度能否再次降低
4.书写程序时，一定要认真，坚决防止出现逻辑错误，并根据程序具体分析可能的极端情况，处理好边界，并自己进行用例测试，以验证程序。

节点的定义如下：

```
typedef struct list {
int key;
struct list *next;
}list;
```

(1)已知链表的头结点head,写一个函数把这个链表逆序 ( Intel)

```
list * reverse(list * head){
list * h = head;
list * new_head = NULL,*temp;
if(h==NULL) return h;//如果是空链表
do{
   temp = h;
   h = h -> next;
   temp -> next = new_head;
   new_head = temp;
}while(h != NULL && h != head)//检测是循环链表
return new_head;
}
```

其实还要注意一点，链表内部是否包含小环。

(2)已知两个链表head1 和head2 各自有序，请把它们合并成一个链表依然有序。(保留所有结点，即便大小相同）

```
list * merge (list *list1_head, list *list2_head){
```

```
}
```

其实需要问一下，head1 head2 是否都是从大到小，这点一定要明确，不能默认两个是相同的规格排序。

(3)已知两个链表head1 和head2 各自有序，请把它们合并成一个链表依然有序，这次要求用递归方法进行。 (Autodesk)

```
list * merge (list *list1_head, list *list2_head){
   list * res;
   if(list1_head == NULL) return list2_head;
   if(list2_head == NULL) return list1_head;
   if(list1_head->key > list2_head->key){
      res = list1_head;
      res->next = merge(list1_head->next,list2_head);

   }else{
      res = list2_head;
      res->next = merge(list1_head,list2_head->next);

   }
   return res;
}
```

(4)写一个程序，计算链表的长度

```
unsigned int list_len(list *head){
  unsigned int len = 0;
  list * h = head;
  if(h == NULL)return 0;
  d0{
      len++;
      h = h->next;
  }while(h != NULL && h != head)
  return len;
}
```

如果是循环链表？

(5) 有一个链表L,其每个节点有 2 个指针，一个指针next指向链表的下个节点，

另一个random随机指向链表中的任一个节点，可能是自己或者为空，写一个程序，要求复制这个链表的结构并分析其复杂性。

这个题目的思路很巧妙。当然简单的方法，可以利用一个map或者hash，将链表的 random指针的指向，保存起来。这样有O(n)存储空间的浪费，时间基本可以接近O(n).

实际上可以这样来做：我们将新的链表节点，插入到原来链表节点当中，并且修改原来的链表的random指针，使得该指针由我们现在的新节点所有。实际上形成下面这样一种结构，同时将原来o的 random指针的值，复制给它后面的现在的@的random指针，该结构如下：

o->@->o->@->o->@->NULL

现在可以利用@拥有的random指针方便的找到它真正的random指针。因为原来的@的random指针指向o的random指针，只要把让@->random=@->random->next,random就是真正的那个指针了，然后我们再把@从这个链表中删除。

(6)给你一个单向链表的头指针，可能最后不是NULL终止，而是循环链表。题目问你怎么找出这个链表循环部分的第一个节点。

这个原来的判断链表环的扩展，需要求出环的开始点。只要稍微对原来的方法进行扩展，也就可以解决这个问题。使用两个指针一个步长为 1，另一个步长为 2，如果第二个指针可以追上第一个指针，则说明有环。这样我们首先可以求出环的长度L，然后在设两个步长为 1 的指针，让他们间距为L，这样第一个相等的地方，就是环的起点。

还有一个更好的方法，就是在步长为 1，步长为 2 指针相遇后，再从头开始一个步长为 1 的指针，然后开始步长为 1 的指针继续行进，当这两个指针相遇的地方就是环的起点。

(7)（华为面试题）找出单向链表中中间结点
这个可以借鉴上面的方法，利用步长为 1，步长为 2 指针，当步长为 2 的指针到达末尾时，指针 1 就刚好达到中间。

(8)题目：给定链表的头指针和一个结点指针，在 O(1)时间删除该结点。链表结点的定义如下：

```
struct ListNode
{
    int         m_nKey;
    ListNode*   m_pNext;
```

};
函数的声明如下：
void DeleteNode(ListNode* pListHead, ListNode* pToBeDeleted);

实际上我们可以将该节点的下一个节点的值copy到这个节点，然后删除下一个节点。这样实际上就将一个不能处理的情况，通过一种技巧转化成了我们可以处理的正常情况。

但是还需要注意下面这个边界情况：如果要删除的是尾指针？这就意味着当前节点没有下一个节点。
通过这点我们也可以发现，如果认真考虑各个因素，思路清晰，还是很容易发现各种异常情况的，保证严谨的思考。

(9)如何检查一个单向链表上是否有环?

见(6)

(10)查找链表中倒数第k个结点

这个可以参考(7),我们可以设置步长相同，间距为k的指针，其特点在当内存无法存放所有元素，需要从硬盘读取时，性能卓越，k在可以接受的情况下可以一次读取到内存，大大提高了效率。

但是仍需要注意考虑边界情况：
（细节，如，假如链表的长度不足m，它就不存在m个元素，因此必须在两个指针的出发阶段检查最先出发的当前指针是否已经到了链表尾。）

(11) 题目：两个单向链表，开头结点不一样，在中间某处开始，结点一样了,找出它们的第一个公共结点。

这个问题，可以通过判断最后一个节点是否相同，判断是否相交。如果相交，可以统计两个链表的长度，设一个为a，一个为b，然后这样再搞两个指针，一个先走|a-b|步，另一个再走，当它们相等时就是第一个公共结点。

再附三个有意思的小题，虽然不是链表相关的。

1.如何判断一个整数是不是完全平方数 (不允许开方). 单调函数可以二分查找。
2.不知长度的日志文件, 如何等概率选出 100 条。
3.一个唱片, 可以被分成 8 个大小一样的小扇形. 要求给每个扇形涂上红色或

者蓝色，使得唱片旋转起来以后，按照顺序读出颜色序列，就能判断唱片是顺时针旋转还是逆时针。

转载请注明出处：http://bbs.xjtu.edu.cn 兵马俑bbs-算法与数据结构版
作者phylips@bmy

--
如果可以
愿把这生命燃烧
只留下星星的传说
悲伤而让人怀念
==
※ 来源:.兵马俑BBS http://bbs.xjtu.edu.cn [FROM: 219.224.191.247]