百度校园招聘笔试面试题

**2013年**

一：简答题（30）  
1：数据库以及线程发生死锁的原理及必要条件，如何避免死锁  
答：  
产生死锁的原因主要是：  
（1） 因为系统资源不足。  
（2） 进程运行推进的顺序不合适。  
（3） 资源分配不当等。  
产生死锁的四个必要条件：  
（1）互斥条件：一个资源每次只能被一个进程使用。  
（2）请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。  
（3）不剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。  
（4）循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

避免死锁：  
死锁的预防是通过破坏产生条件来阻止死锁的产生，但这种方法破坏了系统的并行性和并发性。  
死锁产生的前三个条件是死锁产生的必要条件，也就是说要产生死锁必须具备的条件，而不是存在这3个条件就一定产生死锁，那么只要在逻辑上回避了第四个条件就可以避免死锁。  
避免死锁采用的是允许前三个条件存在，但通过合理的资源分配算法来确保永远不会形成环形等待的封闭进程链，从而避免死锁。该方法支持多个进程的并行执行，为了避免死锁，系统动态的确定是否分配一个资源给请求的进程。  
预防死锁：具体的做法是破坏产生死锁的四个必要条件之一

2：面向对象的三个基本元素，五个基本原则  
答：  
三个基本元素：  
封装  
继承  
多态  
五个基本原则：  
单一职责原则（Single-Resposibility Principle）:一个类，最好只做一件事，只有一个引起它的变化。单一职责原则可以看做是低耦合、高内聚在面向对象原则上的引申，将职责定义为引起变化的原因，以提高内聚性来减少引起变化的原因。  
开放封闭原则（Open-Closed principle）:软件实体应该是可扩展的，而不可修改的。也就是，对扩展开放，对修改封闭的。  
Liskov替换原则（Liskov-Substituion Principle）:子类必须能够替换其基类。这一思想体现为对继承机制的约束规范，只有子类能够替换基类时，才能保证系统在运行期内识别子类，这是保证继承复用的基础。  
依赖倒置原则（Dependecy-Inversion Principle）:依赖于抽象。具体而言就是高层模块不依赖于底层模块，二者都同依赖于抽象；抽象不依赖于具体，具体依赖于抽象。  
接口隔离原则（Interface-Segregation Principle）:使用多个小的专门的接口，而不要使用一个大的总接口。  
3：windows内存管理的机制以及优缺点  
答：  
 分页存储管理基本思想：

用户程序的地址空间被划分成若干固定大小的区域，称为“页”，相应地，内存空间分成若干个物理块，页和块的大小相等。可将用户程序的任一页放在内存的任一块中，实现了离散分配。

分段存储管理基本思想：

将用户程序地址空间分成若干个大小不等的段，每段可以定义一组相对完整的逻辑信息。存储分配时，以段为单位，段与段在内存中可以不相邻接，也实现了离散分配。

段页式存储管理基本思想：

分页系统能有效地提高内存的利用率，而分段系统能反映程序的逻辑结构，便于段的共享与保护，将分页与分段两种存储方式结合起来，就形成了段页式存储管理方式。

在段页式存储管理系统中，作业的地址空间首先被分成若干个逻辑分段，每段都有自己的段号，然后再将每段分成若干个大小相等的页。对于主存空间也分成大小相等的页，主存的分配以页为单位。

段页式系统中，作业的地址结构包含三部分的内容：段号      页号      页内位移量

程序员按照分段系统的地址结构将地址分为段号与段内位移量，地址变换机构将段内位移量分解为页号和页内位移量。

为实现段页式存储管理，系统应为每个进程设置一个段表，包括每段的段号，该段的页表始址和页表长度。每个段有自己的页表，记录段中的每一页的页号和存放在主存中的物理块号。

二：程序设计题（40）  
   
1：公司里面有1001个员工，现在要在公司里面找到最好的羽毛球选手，也就是第一名，每个人都必须参赛，问至少要比赛多少次才能够找到最好的羽毛球员工。  
答：两两比赛，分成500组剩下一人，类似于归并排序的方式，比出冠军后，让冠军之间再比，主要是要想想多余的那一个选手如何处理，必然要在第一次决出冠军后加入比赛组。  
   
2：现在有100个灯泡，每个灯泡都是关着的，第一趟把所有的灯泡灯泡打开，第二趟把偶数位的灯泡制反（也就是开了的关掉，关了的打开），第三趟让第3,6,9....的灯泡制反.......第100趟让第100个灯泡制反，问经过一百趟以后有多少灯泡亮着  
答：   
1．对于每盏灯，拉动的次数是奇数时，灯就是亮着的，拉动的次数是偶数时，灯就是关着的。  
2．每盏灯拉动的次数与它的编号所含约数的个数有关，它的编号有几个约数，这盏灯就被拉动几次。  
3．1——100这100个数中有哪几个数，约数的个数是奇数。我们知道一个数的约数都是成对出现的，只有完全平方数约数的个数才是奇数个。

所以这100盏灯中有10盏灯是亮着的。  
它们的编号分别是： 1、4、9、16、25、36、49、64、81、100。

 3：有20个数组，每个数组有500个元素，并且是有序排列好的，现在在这20\*500个数中找出排名前500的数  
答：TOP-K问题，用个数为K的最小堆来解决

4. 字符串左移，void \*pszStringRotate(char \*pszString, intnCharsRotate),比如ABCDEFG，移3位变DEFGABC，要求空间复杂度O（1），时间复杂度O（n）

**2012年**

1、给一个单词a，如果通过交换单词中字母的顺序可以得到另外的单词b，那么b是a的兄弟单词，比如的单词army和mary互为兄弟单词。  
现在要给出一种解决方案，对于用户输入的单词，根据给定的字典找出输入单词有哪些兄弟单词。请具体说明数据结构和查询流程，要求时间和空间效率尽可能地高。  
字典树的典型应用，一般情况下，字典树的结构都是采用26叉树进行组织的，每个节点对应一个字母，查找的时候，就是一个字母一个字母的进行匹配，算法的时间复杂度就是单词的长度n，效率很高。因此这个题目可以定义一个字典树作为数据结构来查询的，时间效率会很高，这样就转化为在一棵字典树中查找兄弟单词，只要在字典树中的前缀中在存储一个vector结构的容器，这样查找起来就是常数级的时间复杂度了，效率很高的。。  
数据结构可以定义如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/7542774)

1. **struct** word
2. {
3. vector<string> brother;    // 用于保存每个单词的兄弟单词
4. word \*next[26];            // 字典树中每个节点代表一个字符，并指向下一个字符
5. };

如上述数据结构所示，字典树的建立是在预处理阶段完成的，首先根据字典中的单词来建立字典树，建立的时候，需要稍微特殊处理一下，就是比如pots、stop和tops互为兄弟单词，那么在字典中按照首字母顺序的话，应该先遇到pots单词，那么我首先对其进行排序，结果是opts，那么字典树中就分别建立4个节点，分别为o->p->t->s，当然这个是不同层次的，在节点s处的vector容器brother中添加单词pots，遇到stop的时候，同样的方法，排序是opts，此时发现这4个节点已经建立了，那么只需要在第四个节点s处的vector容器brother中添加单词stop，tops单词的处理方法是同样的。  
     这样建立完字典树后，查询兄弟单词的效率就会很高了，比哈希的效率还要高；查到tops的兄弟的单词的时候，首先排序，那么就是opts，然后在字典树中查找opts，在s处将其vector容器brother中的的单词输出就是tops的所有兄弟单词。  
2、系统中维护了若干数据项，我们对数据项的分类可以分为三级，首先我们按照一级分类方法将数据项分为A、B、C......若干类别，每个一级分类方法产生的类别又可以按照二级分类方法分为a、b、c......若干子类别，同样，二级分类方法产生的类别又可以按照是三级分类方法分为i、ii、iii......若干子类别，每个三级分类方法产生的子类别中的数据项从1开始编号。我们需要对每个数据项输出日志，日志的形式是key\_value对，写入日志的时候，用户提供三级类别名称、数据项编号和日志的key，共五个key值，例如，write\_log（A,a,i,1,key1），获取日志的时候，用户提供三级类别名称、数据项编号，共四个key值，返回对应的所有的key\_value对，例如get\_log（A,a,i,1,key1），  
请描述一种数据结构来存储这些日志，并计算出写入日志和读出日志的时间复杂度。  
  
3、C和C++中如何动态分配和释放内存？他们的区别是什么？  
malloc/free和new/delete的区别，请参考这里<http://blog.csdn.net/hackbuteer1/article/details/6789164>

4、数组al[0,mid-1]和al[mid,num-1]是各自有序的，对数组al[0,num-1]的两个子有序段进行merge，得到al[0,num-1]整体有序。要求空间复杂度为O(1)。注：al[i]元素是支持'<'运算符的。

**[cpp]** [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/7542774)

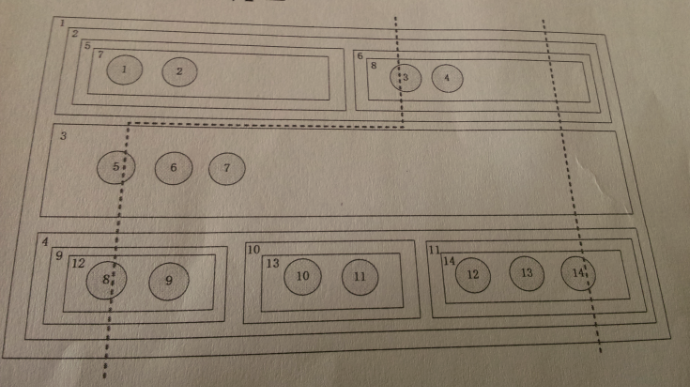
1. /\*
2. 数组a[begin, mid] 和 a[mid+1, end]是各自有序的，对两个子段进行Merge得到a[begin , end]的有序数组。 要求空间复杂度为O(1)
3. 方案：
4. 1、两个有序段位A和B，A在前，B紧接在A后面，找到A的第一个大于B[0]的数A[i]， A[0...i-1]相当于merge后的有效段，在B中找到第一个大于A[i]的数B[j]，
5. 对数组A[i...j-1]循环右移j-k位，使A[i...j-1]数组的前面部分有序
6. 2、如此循环merge
7. 3、循环右移通过先子段反转再整体反转的方式进行，复杂度是O(L), L是需要循环移动的子段的长度
8. \*/
9. #include<iostream>
10. **using** **namespace** std;
12. **void** Reverse(**int** \*a , **int** begin , **int** end )   //反转
13. {
14. **for**(; begin < end; begin++ , end--)
15. swap(a[begin] , a[end]);
16. }
17. **void** RotateRight(**int** \*a , **int** begin , **int** end , **int** k)    //循环右移
18. {
19. **int** len = end - begin + 1;  //数组的长度
20. k %= len;
21. Reverse(a , begin , end - k);
22. Reverse(a , end - k + 1 , end);
23. Reverse(a , begin , end);
24. }
26. // 将有序数组a[begin...mid] 和 a[mid+1...end] 进行归并排序
27. **void** Merge(**int** \*a , **int** begin , **int** end )
28. {
29. **int** i , j , k;
30. i = begin;
31. j = 1 + ((begin + end)>>1);    //位运算的优先级比较低，外面需要加一个括号，刚开始忘记添加括号，导致错了很多次
32. **while**(i <= end && j <= end && i<j)
33. {
34. **while**(i <= end && a[i] < a[j])
35. i++;
36. k = j;   //暂时保存指针j的位置
37. **while**(j <= end && a[j] < a[i])
38. j++;
39. **if**(j > k)
40. RotateRight(a , i , j-1 , j-k);   //数组a[i...j-1]循环右移j-k次
41. i += (j-k+1);  //第一个指针往后移动，因为循环右移后，数组a[i....i+j-k]是有序的
42. }
43. }
45. **void** MergeSort(**int** \*a , **int** begin , **int** end )
46. {
47. **if**(begin == end)
48. **return** ;
49. **int** mid = (begin + end)>>1;
50. MergeSort(a , begin , mid);   //递归地将a[begin...mid] 归并为有序的数组
51. MergeSort(a , mid+1 , end);   //递归地将a[mid+1...end] 归并为有序的数组
52. Merge(a , begin , end);       //将有序数组a[begin...mid] 和 a[mid+1...end] 进行归并排序
53. }
55. **int** main(**void**)
56. {
57. **int** n , i , a[20];
58. **while**(cin>>n)
59. {
60. **for**(i = 0 ; i < n ; ++i)
61. cin>>a[i];
62. MergeSort(a , 0 , n - 1);
63. **for**(i = 0 ; i < n ; ++i)
64. cout<<a[i]<<" ";
65. cout<<endl;
66. }
67. **return** 0;
68. }

5、线程和进程的区别及联系？如何理解“线程安全”问题？

答案：进程和线程都是由操作系统所体会的程序运行的基本单元，系统利用该基本单元实现系统对应用的并发性。  
1、简而言之,一个程序至少有一个进程,一个进程至少有一个线程.  
2、线程的划分尺度小于进程，使得多线程程序的并发性高。  
3、另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。  
4、线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。  
5、从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

算法与程序设计一、  
网页爬虫在抓取网页时，从指定的URL站点入口开始爬取这个站点上的所有URL link，抓取到下一级link对应的页面后，同样对页面上的link进行抓取从而完成深度遍历。为简化问题，我们假设每个页面上至多只有一个link，如从www.baidu.com/a.html链接到  
www.baidu.com/b.html再链到www.baidu.com/x.html，当爬虫抓取到某个页面时，有可能再链回www.baidu.com/b.html，也有可能爬取到一个不带任何link的终极页面。当抓取到相同的URL或不包含任何link的终极页面时即完成爬取。爬虫在抓取到这些页面后建立一个单向链表，用来记录抓取到的页面，如：a.html->b.html->x.html...->NULL。  
问：对于爬虫分别从www.baidu.com/x1.html和www.baidu.com/x2.html两个入口开始获得两个单向链表，得到这两个单向链表后，如何判断他们是否抓取到了相同的URL？（假设页面URL上百亿，存储资源有限，无法用hash方法判断是否包含相同的URL）  
请先描述相应的算法，再给出相应的代码实现。（只需给出判断方法代码，无需爬虫代码）  
两个单向链表的相交问题。  
算法与程序设计二、  
4、有一种结构如下图所示，它由层的嵌套组成，一个父层中只能包含垂直方向上或者是水平方向上并列的层，例如，层1可以包含2、3、4三个垂直方向上的层，层2可以包含5和6两个水平方向的层，在空层中可以包含数据节点，所谓的空层是指不包含子层的层，每个空层可以包含若干个数据节点，也可以一个都不包含。  
在这种结构上面，我们从垂直方向上划一条线，我们约定每一个子层中我们只能经过一个数据节点，在这种情况下，每条线可以经过多个数据节点，也可以不经过任何数据节点，例如，线1经过了3、5、8三个数据节点，线2只经过了14个数据节点。  
（1）给出函数，实现判断两个数据节点，是否可能同时被线划中，给出具体的代码。

（2）给出函数，输出所有一条线可以划中的数据节点序列， 可以给出伪代码实现。



思路：（1）判断两个数所属的同一层次的相同矩形框的下一层次矩形框是水平排列的还是垂直排列的，垂直排列在能在一条线上，水平排列则不能。  
（2）用回溯算法求出所有在一条直线上的字符串，用两字符串是否在同一直线上进行剪枝操作。  
系统设计题  
1、相信大家都使用过百度搜索框的suggestion功能，百度搜索框中的suggestion提示功能如何实现？请给出实现思路和主要的数据结构、算法。有什么优化思路可以使得时间和空间效率最高？  
应用字典树来求前缀和TOP K对热词进行统计排序  
2、两个200G大小的文件A和B，AB文件里内容均为无序的一行一个正整数字（不超过2^63），请设计方案，输出两个文件中均出现过的数字，使用一台内存不超过16G、磁盘充足的机器。  
方案中指明使用java编程时使用到的关键工具类，以及为什么?

**2011年**

 一、算法设计  
1、设rand（s，t）返回[s,t]之间的随机小数，利用该函数在一个半径为R的圆内找随机n个点，并给出时间复杂度分析。

思路：这个使用数学中的极坐标来解决，先调用[s1，t1]随机产生一个数r，归一化后乘以半径，得到R\*（r-s1）/（t1-s1），然后在调用[s2，t2]随机产生一个数a，归一化后得到角度：360\*（a-s2）/（t2-s2）

2、为分析用户行为，系统常需存储用户的一些query，但因query非常多，故系统不能全存，设系统每天只存m个query，现设计一个算法，对用户请求的query进行随机选择m个，请给一个方案，使得每个query被抽中的概率相等，并分析之，注意：不到最后一刻，并不知用户的总请求量。

思路：如果用户查询的数量小于m，那么直接就存起来。如果用户查询的数量大于m，假设为m+i，那么在1-----m+i之间随机产生一个数，如果选择的是前面m条查询进行存取，那么概率为m/（m+i），如果选择的是后面i条记录中的查询，那么用这个记录来替换前面m条查询记录的概率为m/（m+i）\*（1-1/m）=(m-1)/(m+i)，当查询记录量很大的时候，m/（m+i）== (m-1)/(m+i)，所以每个query被抽中的概率是相等的。

3、C++ STL中vector的相关问题：  
    （1）、调用push\_back时，其内部的内存分配是如何进行的？  
    （2）、调用clear时，内部是如何具体实现的？若想将其内存释放，该如何操作？

vector的工作原理是系统预先分配一块CAPACITY大小的空间，当插入的数据超过这个空间的时候，这块空间会让某种方式扩展，但是你删除数据的时候，它却不会缩小。  
  vector为了防止大量分配连续内存的开销，保持一块默认的尺寸的内存，clear只是清数据了，未清内存，因为vector的capacity容量未变化，系统维护一个的默认值。

有什么方法可以释放掉vector中占用的全部内存呢?

标准的解决方法如下  
template < class T >  
void ClearVector( vector< T >& vt )  
{  
vector< T > vtTemp;  
veTemp.swap( vt );  
}

  事实上，vector根本就不管内存，它只是负责向内存管理框架acquire/release内存，内存管理框架如果发现内存不够了，就malloc，但是当vector释放资源的时候(比如destruct), stl根本就不调用free以减少内存，因为内存分配在stl的底层：stl假定如果你需要更多的资源就代表你以后也可能需要这么多资源(你的list, hashmap也是用这些内存)，所以就没必要不停地malloc/free。如果是这个逻辑的话这可能是个trade-off

  一般的STL内存管理器allocator都是用内存池来管理内存的，所以某个容器申请内存或释放内存都只是影响到内存池的剩余内存量，而不是真的把内存归还给系统。这样做一是为了避免内存碎片，二是提高了内存申请和释放的效率——不用每次都在系统内存里寻找一番。

二、系统设计  
正常用户端每分钟最多发一个请求至服务端，服务端需做一个异常客户端行为的过滤系统，设服务器在某一刻收到客户端A的一个请求，则1分钟内的客户端任何其它请求都需要被过滤，现知每一客户端都有一个IPv6地址可作为其ID，客户端个数太多，以至于无法全部放到单台服务器的内存hash表中，现需简单设计一个系统，使用支持高效的过滤，可使用多台机器，但要求使用的机器越少越好，请将关键的设计和思想用图表和代码表现出来。

三、求一个全排列函数：  
如p([1,2,3])输出：  
[123]、[132]、[213]、[231]、[321]、[312]  
 求一个组合函数。

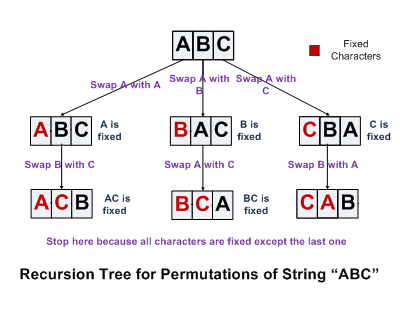
方法1：依次从字符串中取出一个字符作为最终排列的第一个字符，对剩余字符组成的字符串生成全排列，最终结果为取出的字符和剩余子串全排列的组合。

**[cpp]** [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/6878627)

1. #include <iostream>
2. #include <string>
3. **using** **namespace** std;
5. **void** permute1(string prefix, string str)
6. {
7. **if**(str.length() == 0)
8. cout << prefix << endl;
9. **else**
10. {
11. **for**(**int** i = 0; i < str.length(); i++)
12. permute1(prefix+str[i], str.substr(0,i)+str.substr(i+1,str.length()));
13. }
14. }
16. **void** permute1(string s)
17. {
18. permute1("",s);
19. }
21. **int** main(**void**)
22. {
23. //method1, unable to remove duplicate permutations.
24. permute1("abc");
25. **return** 0;
26. }

优点：该方法易于理解，但无法移除重复的排列，如：s="ABA"，会生成两个“AAB”。

方法2：利用交换的思想，具体见实例，但该方法不如方法1容易理解。



我们以三个字符abc为例来分析一下求字符串排列的过程。首先我们固定第一个字符a，求后面两个字符bc的排列。当两个字符bc的排列求好之后，我们把第一个字符a和后面的b交换，得到bac，接着我们固定第一个字符b，求后面两个字符ac的排列。现在是把c放到第一位置的时候了。记住前面我们已经把原先的第一个字符a和后面的b做了交换，为了保证这次c仍然是和原先处在第一位置的a交换，我们在拿c和第一个字符交换之前，先要把b和a交换回来。在交换b和a之后，再拿c和处在第一位置的a进行交换，得到cba。我们再次固定第一个字符c，求后面两个字符b、a的排列。

既然我们已经知道怎么求三个字符的排列，那么固定第一个字符之后求后面两个字符的排列，就是典型的递归思路了。

基于前面的分析，我们可以得到如下的参考代码：

**[cpp]** [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/6878627)

1. **void** Permutation(**char**\* pStr, **char**\* pBegin)
2. {
3. assert(pStr && pBegin);
4. //if(!pStr || !pBegin)
5. //return ;
6. **if**(\*pBegin == '\0')
7. printf("%s\n",pStr);
8. **else**
9. {
10. **char** temp;
11. **for**(**char**\* pCh = pBegin; \*pCh != '\0'; pCh++)
12. {
13. **if**(pCh != pBegin && \*pCh == \*pBegin)   //为避免生成重复排列，当不同位置的字符相同时不再交换
14. **continue**;
15. temp = \*pCh;
16. \*pCh = \*pBegin;
17. \*pBegin = temp;
19. Permutation(pStr, pBegin+1);
21. temp = \*pCh;
22. \*pCh = \*pBegin;
23. \*pBegin = temp;
24. }
25. }
26. }
28. **int** main(**void**)
29. {
30. **char** str[] = "aba";
31. Permutation(str,str);
32. **return** 0;
33. }

如p([1,2,3])输出：  
[1]、[2]、[3]、[1,2]、[2,3]、[1,3]、[1,2,3]  
这两问可以用伪代码。

**2011面试题**

1、进程切换需要注意哪些问题？

保存处理器PC寄存器的值到被中止进程的私有堆栈；      保存处理器PSW寄存器的值到被中止进程的私有堆栈；    保存处理器SP寄存器的值到被中止进程的进程控制块；

保存处理器其他寄存器的值到被中止进程的私有堆栈；     自待运行进程的进程控制块取SP值并存入处理器的寄存器SP；    自待运行进程的私有堆栈恢复处理器各寄存器的值；

自待运行进程的私有堆栈中弹出PSW值并送入处理器的PSW；     自待运行进程的私有堆栈中弹出PC值并送入处理器的PC。

2、输入一个升序数组，然后在数组中快速寻找两个数字，其和等于一个给定的值。

这个编程之美上面有这个题目的，很简单的，用两个指针一个指向数组前面，一个指向数组的后面，遍历一遍就可以了。

3、有一个名人和很多平民在一块，平民都认识这个名人，但是这个名人不认识任何一个平民，任意两个平民之间是否认识是未知的，请设计一个算法，快速找个这个人中的那个名人。  已知已经实现了一个函数  bool know(int a,int b) 这个函数返回true的时候，表明a认识b，返回false的时候表明a不认识b。

思路：首先将n个人分为n/2组，每一组有2个人，然后每个组的两个人调用这个know函数，假设为know（a，b），返回true的时候说明a认识b，则a肯定不是名人，a可以排除掉了，依次类推，每个组都调用这个函数依次，那么n个人中就有n/2个人被排除掉了，数据规模将为n/2。同理在剩下的n/2个人中在使用这个方法，那么规模就会将为n/4，这样所有的遍历次数为n/2+n/4+n/8+........ 这个一个等比数列，时间复杂度为o（n）。

4、有一类数组，例如书序[1,2,3,4,6,8,9,4,8,11,18,19,100] 前半部分是是一个递增数组，后面一个还是递增数组，但整个数组不是递增数组，那么怎么最快的找出其中一个数？

**[cpp]** [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/6882705)

1. #include <iostream>
2. **using** **namespace** std;
4. **int** binary\_search(**int**\* a, **int** low, **int** high, **int** goal)    //二分查找
5. {
6. **while**(low <= high)
7. {
8. **int** middle = (low+high)>>1;    //(low+high)/2
9. **if**(a[middle] == goal)
10. **return** middle;
11. //在右半边
12. **else** **if**(a[middle] < goal)
13. low = middle + 1;
14. //在左半边
15. **else**
16. high = middle - 1;
17. }
18. **return** -1;
19. }
20. **void** getNum(**int** \*a, **int** len, **int** goal)
21. {
22. **int** i, index;
23. **for**(i = 0; i < len-1; i++)
24. {
25. **if**(a[i] > a[i+1])     //找到前、后两个数组的分界点
26. **break**;
27. }
28. **if**(a[i] >= goal)          //对前面数组进行二分查找
29. {
30. index = binary\_search(a, 0, i, goal);
31. printf("%d\n",index);
32. }
33. **if**(a[i+1] <= goal)         //对后面数组进行二分查找
34. {
35. index = binary\_search(a+i+1, 0, len-i-2, goal);
36. **if**(index != -1)
37. index += (i+1);    //后面的那个数组相对于前面数组的偏移量为i+1
38. printf("%d\n",index);
39. }
40. }
42. **int** main(**void**)
43. {
44. **int** a[]={1,2,3,4,6,8,9,4,8,11,18,19,100};
45. **int** len = 13, goal = 4;
46. getNum(a,len,goal);
47. **return** 0;
48. }

5、判断一个自然数是否是某个数的平方。当然不能使用开方运算。

方法1：  
遍历从1到N的数字，求取平方并和N进行比较。  
如果平方小于N，则继续遍历；如果等于N，则成功退出；如果大于N，则失败退出。  
复杂度为O(n^0.5)。

方法2：  
使用二分查找法，对1到N之间的数字进行判断。  
复杂度为O(log n)。

方法3：  
由于  
(n+1)^2  
=n^2 + 2n + 1，  
= ...  
= 1 + (2\*1 + 1) + (2\*2 + 1) + ... + (2\*n + 1)  
注意到这些项构成了等差数列（每项之间相差2）。  
所以我们可以比较 N-1， N - 1 - 3， N - 1 - 3 - 5 ... 和0的关系。  
如果大于0，则继续减；如果等于0，则成功退出；如果小于 0，则失败退出。  
复杂度为O(n^0.5)。不过方法3中利用加减法替换掉了方法1中的乘法，所以速度会更快些。

例如：3^2 = 9 = 1 + 2\*1+1 + 2\*2+1 = 1 + 3 + 5

4^2 = 16 = 1 + 2\*1 + 1 + 2\*2+1  + 2\*3+1

**[cpp]** [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/6882705)

1. **int** square(**int** n)
2. {
3. **int** i = 1;
4. n = n - i;
5. **while**( n > 0 )
6. {
7. i += 2;
8. n -= i;
9. }
10. **if**( n == 0 )        //是某个数的平方
11. **return** 1;
12. **else**                //不是某个数的平方
13. **return** 0;
14. }

 最新笔试题

1、实现一个函数，对一个正整数n，算得到1需要的最少操作次数。操作规则为：如果n为偶数，将其除以2；如果n为奇数，可以加1或减1；一直处理下去。

例子：  
func(7) = 4，可以证明最少需要4次运算  
n = 7  
n-1 6  
n/2 3  
n-1 2  
n/2 1  
要求：实现函数(实现尽可能高效) int func(unsign int n)；n为输入，返回最小的运算次数。给出思路(文字描述)，完成代码，并分析你算法的时间复杂度。  
答：

**[cpp]** [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/7348968)

1. **int** func(unsigned **int** n)
2. {
3. **if**(n == 1)
4. **return** 0;
5. **if**(n % 2 == 0)
6. **return** 1 + func(n/2);
7. **int** x = func(n + 1);
8. **int** y = func(n - 1);
9. **if**(x > y)
10. **return** y+1;
11. **else**
12. **return** x+1;
13. }

假设n表示成二进制有x bit，可以看出计算复杂度为O(2^x)，也就是O(n)。  
将n转换到二进制空间来看（比如7为111，6为110）：  
- 如果最后一位是0，则对应于偶数，直接进行除2操作。  
- 如果最后一位是1，情况则有些复杂。  
\*\*如果最后几位是???01，则有可能为???001，???1111101。在第一种情况下，显然应该-1；在第二种情况下-1和+1最终需要的步数相同。所以在???01的情况下，应该选择-1操作。  
\*\*如果最后几位是???011，则有可能为???0011，???11111011。在第一种情况下，+1和-1最终需要的步数相同；在第二种情况下+1步数更少些。所以在???011的情况下，应该选择+1操作。  
\*\*如果最后有更多的连续1，也应该选择+1操作。

如果最后剩下的各位都是1，则有11时应该选择-1；111时+1和-1相同；1111时应选择+1；大于四个1时也应该选择+1；

**[cpp]** [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/7348968)

1. **int** func(unsigned **int** n)
2. {
3. **if**(n == 1)
4. **return** 0;
5. **if**(n % 2 == 0)
6. **return** 1 + func(n/2);
7. **if**(n == 3)
8. **return** 2;
9. **if**(n&2)
10. **return** 1 + func(n+1);
11. **else**
12. **return** 1 + func(n-1);
13. }

由以上的分析可知，奇数的时候加1或减1，完全取决于二进制的后两位，如果后两位是10、00那么肯定是偶数，选择除以2，如果后两位是01、11，那么选择结果会不一样的，如果是\*\*\*\*\*01，那么选择减1，如果是\*\*\*\*\*11，那么选择加1，特殊情况是就是n是3的时候，选择减1操作。  
非递归代码如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/7348968)

1. // 非递归写法
2. **int** func(**int** n)
3. {
4. **int** count = 0;
5. **while**(n > 1)
6. {
7. **if**(n % 2 == 0)
8. n >>= 1;
9. **else** **if**(n == 3)
10. n--;
11. **else**
12. {
13. **if**(n&2)      // 二进制是\*\*\*\*\*\*11时
14. n++;
15. **else**         // 二进制是\*\*\*\*\*\*01时
16. n--;
17. }
18. count++;
19. }
20. **return** count;
21. }

另外一种写法如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/7348968)

1. // 非递归写法
2. **int** func(**int** n)
3. {
4. **int** count = 0;
5. **while**(n > 1)
6. {
7. **if**(n % 2 == 0)            // n % 4等于0或2
8. n >>= 1;
9. **else** **if**(n == 3)
10. n--;
11. **else**
12. n += (n % 4 - 2);     // n % 4等于1或3
13. count++;
14. }
15. **return** count;
16. }

2、找到满足条件的数组  
给定函数d(n)=n+n的各位之和，n为正整数，如d(78)=78+7+8=93。这样这个函数可以看成一个生成器，如93可以看成由78生成。  
定义数A：数A找不到一个数B可以由d(B)=A，即A不能由其他数生成。现在要写程序，找出1至10000里的所有符合数A定义的数。  
回答：  
申请一个长度为10000的bool数组，每个元素代表对应的值是否可以有其它数生成。开始时将数组中的值都初始化为false。  
由于大于10000的数的生成数必定大于10000，所以我们只需遍历1到10000中的数，计算生成数，并将bool数组中对应的值设置为true，表示这个数可以有其它数生成。  
最后bool数组中值为false的位置对应的整数就是不能由其它数生成的。  
3、一个大的含有50M个URL的记录，一个小的含有500个URL的记录，找出两个记录里相同的URL。

回答：  
首先使用包含500个url的文件创建一个hash\_set。  
然后遍历50M的url记录，如果url在hash\_set中，则输出此url并从hash\_set中删除这个url。  
所有输出的url就是两个记录里相同的url。  
4、海量日志数据，提取出某日访问百度次数最多的那个IP。  
回答：  
如果日志文件足够的大，大到不能完全加载到内存中的话。  
那么可以考虑分而治之的策略，按照IP地址的hash(IP)%1024值，将海量日志存储到1024个小文件中。每个小文件最多包含4M个IP地址。  
对于每个小文件，可以构建一个IP作为key，出现次数作为value的hash\_map，并记录当前出现次数最多的1个IP地址。  
有了1024个小文件中的出现次数最多的IP，我们就可以轻松得到总体上出现次数最多的IP。  
5、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。如何按照query的频度排序？  
回答：  
　1）读取10个文件，按照hash(query)%10的结果将query写到对应的文件中。这样我们就有了10个大小约为1G的文件。任意一个query只会出现在某个文件中。  
　2）对于1）中获得的10个文件，分别进行如下操作  
　　-利用hash\_map（query，query\_count）来统计每个query出现的次数。  
　　-利用堆排序算法对query按照出现次数进行排序。  
　　-将排序好的query输出的文件中。  
　　这样我们就获得了10个文件，每个文件中都是按频率排序好的query。  
　3）对2）中获得的10个文件进行归并排序，并将最终结果输出到文件中。

6、蚂蚁爬杆问题  
　有一根27厘米长的细木杆，在第3厘米，7厘米，11厘米，17厘米，23厘米这五个位置上各有一只蚂蚁，木杆很细，不能同时通过两只蚂蚁，开始时，蚂蚁的头朝向左还是右是任意的，他们只会朝前走或掉头，但不会后退，当两只蚂蚁相遇后，蚂蚁会同时掉头朝反方向走，假设蚂蚁们每秒钟可以走1厘米的距离。求所有蚂蚁都离开木杆的最小时间和最大时间。  
　答案：  
　两只蚂蚁相遇后，各自掉头朝相反方向走。如果我们不考虑每个蚂蚁的具体身份，这和两只蚂蚁相遇后，打个招呼继续向前走没有什么区别。  
　所有蚂蚁都离开木杆的最小时间为  
　max(min(3,27-3),min(7,27-7), min(11,27-11), min(17,27-17),min(23,27-23))=11  
　所有蚂蚁都离开木杆的最大时间为  
　max(max(3,27-3),max(7,27-7), max(11,27-11), max(17,27-17),max(23,27-23))=24

7、当在浏览器中输入一个url后回车，后台发生了什么？比如输入url后，你看到了百度的首页，那么这一切是如何发生的呢？  
回答：  
　　简单来说有以下步骤：  
　　1、查找域名对应的IP地址。这一步会依次查找浏览器缓存，系统缓存，路由器缓存，ISPDNS缓存，根域名服务器。  
　　2、向IP对应的服务器发送请求。  
　　3、服务器响应请求，发回网页内容。  
　　4、浏览器解析网页内容。  
当然，由于网页可能有重定向，或者嵌入了图片，AJAX，其它子网页等等，这4个步骤可能反复进行多次才能将最终页面展示给用户。

8、判断两棵树是否相等，请实现两棵树是否相等的比较，相等返回1，否则返回其他值，并说明算法复杂度。

数据结构为：

**[cpp]** [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/7348968)

1. **typedef** **struct** TreeNode
2. {
3. **char** c;
4. TreeNode \*leftchild;
5. TreeNode \*rightchild;
6. }TreeNode;

函数接口为：int CompTree(TreeNode\* tree1,TreeNode\* tree2);  
注：A、B两棵树相等当且仅当RootA->c==RootB-->c,而且A和B的左右子树相等或者左右互换相等。  
递归方法：

**[cpp]** [view plaincopy](http://blog.csdn.net/hackbuteer1/article/details/7348968)

1. **bool** CompTree(TreeNode \*tree1, TreeNode \*tree2)
2. {
3. **if**(tree1 == NULL && tree2 == NULL)
4. **return** **true**;
5. **if**(tree1 == NULL || tree2 == NULL)
6. **return** **false**;
7. **if**(tree1->c != tree2->c)
8. **return** **false**;
9. **if**( (CompTree(tree1->leftchild, tree2->leftchild) && CompTree(tree1->rightchild, tree2->rightchild)) || CompTree(tree1->leftchild, tree2->rightchild) && CompTree(tree1->rightchild, tree2->leftchild))
10. **return** **true**;
11. }

时间复杂度：  
 在树的第0层，有1个节点，我们会进行1次函数调用；  
 在树的第1层，有2个节点，我们可能会进行4次函数调用；  
 在树的第2层，有4个节点，我们可能会进行16次函数调用；  
 ....  
 在树的第x层，有2^x个节点，我们可能会进行(2^x)^2次函数调用；  
 所以假设总节点数为n，则算法的复杂度为O(n^2)。

**腾讯面试题：求一个论坛的在线人数，假设有一个论坛，其注册ID有两亿个，每个ID从登陆到退出会向一个日志文件中记下登陆时间和退出时间，要求写一个算法统计一天中论坛的用户在线分布，取样粒度为秒。**  
回答：  
　　一天总共有3600\*24=86400秒。  
　　定义一个长度为86400的整数数组intdelta[86400]，每个整数对应这一秒的人数变化值，可能为正也可能为负。开始时将数组元素都初始化为0。  
　　然后依次读入每个用户的登录时间和退出时间，将与登录时间对应的整数值加1，将与退出时间对应的整数值减1。  
　　这样处理一遍后数组中存储了每秒中的人数变化情况。  
　　定义另外一个长度为86400的整数数组intonline\_num[86400]，每个整数对应这一秒的论坛在线人数。  
　　假设一天开始时论坛在线人数为0，则第1秒的人数online\_num[0]=delta[0]。第n+1秒的人数online\_num[n]=online\_num[n-1]+delta[n]。  
　　这样我们就获得了一天中任意时间的在线人数。

9、三个警察和三个囚徒的过河问题

　　三个警察和三个囚徒共同旅行。一条河挡住了去路，河边有一条船，但是每次只能载2人。存在如下的危险：无论在河的哪边，当囚徒人数多于警察的人数时，将有警察被囚徒杀死。问题：请问如何确定渡河方案，才能保证6人安全无损的过河。  
答案：第一次：两囚徒同过，回一囚徒  
第二次：两囚徒同过，回一囚徒  
第三次：两警察同过，回一囚徒一警察（此时对岸还剩下一囚徒一警察，是安全状态）  
第四次：两警察同过，回一囚徒（此时对岸有3个警察，是安全状态）  
第五次：两囚徒同过，回一囚徒  
第六次：两囚徒同过；over

10、从300万字符串中找到最热门的10条  
搜索的输入信息是一个字符串，统计300万输入信息中的最热门的前10条，我们每次输入的一个字符串为不超过255byte，内存使用只有1G。请描述思想，写出算法（c语言），空间和时间复杂度。  
答案：  
　　300万个字符串最多（假设没有重复，都是最大长度）占用内存3M\*1K/4=0.75G。所以可以将所有字符串都存放在内存中进行处理。  
　　可以使用key为字符串（事实上是字符串的hash值），值为字符串出现次数的hash来统计每个每个字符串出现的次数。并用一个长度为10的数组/链表来存储目前出现次数最多的10个字符串。  
　　这样空间和时间的复杂度都是O(n)。  
11、如何找出字典中的兄弟单词。给定一个单词a，如果通过交换单词中字母的顺序可以得到另外的单词b，那么定义b是a的兄弟单词。现在给定一个字典，用户输入一个单词，如何根据字典找出这个单词有多少个兄弟单词？  
答案：  
　　使用hash\_map和链表。  
　　首先定义一个key，使得兄弟单词有相同的key，不是兄弟的单词有不同的key。例如，将单词按字母从小到大重新排序后作为其key，比如bad的key为abd，good的key为dgoo。  
　　使用链表将所有兄弟单词串在一起，hash\_map的key为单词的key，value为链表的起始地址。  
　　开始时，先遍历字典，将每个单词都按照key加入到对应的链表当中。当需要找兄弟单词时，只需求取这个单词的key，然后到hash\_map中找到对应的链表即可。  
　　这样创建hash\_map时时间复杂度为O(n)，查找兄弟单词时时间复杂度是O(1)。  
12、找出数组中出现次数超过一半的数，现在有一个数组，已知一个数出现的次数超过了一半，请用O(n)的复杂度的算法找出这个数。  
答案1：  
　　创建一个hash\_map，key为数组中的数，value为此数出现的次数。遍历一遍数组，用hash\_map统计每个数出现的次数，并用两个值存储目前出现次数最多的数和对应出现的次数。  
　　这样可以做到O(n)的时间复杂度和O(n)的空间复杂度，满足题目的要求。  
　　但是没有利用“一个数出现的次数超过了一半”这个特点。也许算法还有提高的空间。  
答案2：  
　　使用两个变量A和B，其中A存储某个数组中的数，B用来计数。开始时将B初始化为0。  
　　遍历数组，如果B=0，则令A等于当前数，令B等于1；如果当前数与A相同，则B=B+1；如果当前数与A不同，则令B=B-1。遍历结束时，A中的数就是要找的数。  
　　这个算法的时间复杂度是O(n)，空间复杂度为O(1)。

13、找出被修改过的数字  
      n个空间（其中n<1M），存放a到a+n-1的数，位置随机且数字不重复，a为正且未知。现在第一个空间的数被误设置为-1。已经知道被修改的数不是最小的。请找出被修改的数字是多少。  
　　例如：n=6，a=2，原始的串为5,3,7,6,2,4。现在被别人修改为-1,3,7,6,2,4。现在希望找到5。  
回答：  
　　由于修改的数不是最小的，所以遍历第二个空间到最后一个空间可以得到a的值。  
　　a到a+n-1这n个数的和是total=na+(n-1)n/2。  
　　将第二个至最后一个空间的数累加获得sub\_total。  
　　那么被修改的数就是total-sub\_total。

14、设计DNS服务器中cache的数据结构。  
　　要求设计一个DNS的Cache结构，要求能够满足每秒5000以上的查询，满足IP数据的快速插入，查询的速度要快。（题目还给出了一系列的数据，比如：站点数总共为5000万，IP地址有1000万，等等）  
回答：  
　　DNS服务器实现域名到IP地址的转换。  
　　每个域名的平均长度为25个字节（估计值），每个IP为4个字节，所以Cache的每个条目需要大概30个字节。  
　　总共50M个条目，所以需要1.5G个字节的空间。可以放置在内存中。（考虑到每秒5000次操作的限制，也只能放在内存中。）  
　　可以考虑的数据结构包括hash\_map，字典树，红黑树等等。  
15、找出给定字符串对应的序号。  
　　序列Seq=[a,b,…z,aa,ab…az,ba,bb,…bz,…,za,zb,…zz,aaa,…]类似与excel的排列，任意给出一个字符串s=[a-z]+(由a-z字符组成的任意长度字符串），请问s是序列Seq的第几个。  
回答：  
　　注意到每满26个就会向前进一位，类似一个26进制的问题。  
　　比如ab，则位置为26\*1+2；  
　　比如za，则位置为26\*26+1；  
　　比如abc，则位置为26\*26\*1+26\*2+3；  
16、找出第k大的数字所在的位置。写一段程序，找出数组中第k大小的数，输出数所在的位置。例如{2，4，3，4，7}中，第一大的数是7，位置在4。第二大、第三大的数都是4，位置在1、3随便输出哪一个均可。  
答案：  
   先找到第k大的数字，然后再遍历一遍数组找到它的位置。所以题目的难点在于如何最高效的找到第k大的数。  
　我们可以通过快速排序，堆排序等高效的排序算法对数组进行排序，然后找到第k大的数字。这样总体复杂度为O(NlogN)。  
　我们还可以通过二分的思想，找到第k大的数字，而不必对整个数组排序。从数组中随机选一个数t，通过让这个数和其它数比较，我们可以将整个数组分成了两部分并且满足，{x,xx,...,t}<{y,yy,...}。  
　在将数组分成两个数组的过程中，我们还可以记录每个子数组的大小。这样我们就可以确定第k大的数字在哪个子数组中。  
　然后我们继续对包含第k大数字的子数组进行同样的划分，直到找到第k大的数字为止。  
　平均来说，由于每次划分都会使子数组缩小到原来1/2，所以整个过程的复杂度为O(N)。

17、给40亿个不重复的unsigned int的整数，没排过序的，然后再给几个数，如何快速判断这几个数是否在那40亿个数当中?  
答案：  
　　unsigned int的取值范围是0到2^32-1。我们可以申请连续的2^32/8=512M的内存，用每一个bit对应一个unsigned int数字。首先将512M内存都初始化为0，然后每处理一个数字就将其对应的bit设置为1。当需要查询时，直接找到对应bit，看其值是0还是1即可。

18、在一个文件中有10G个整数，乱序排列，要求找出中位数。内存限制为2G。  
回答：  
　　不妨假设10G个整数是64bit的。  
　　2G内存可以存放256M个64bit整数。  
　　我们可以将64bit的整数空间平均分成256M个取值范围，用2G的内存对每个取值范围内出现整数个数进行统计。这样遍历一边10G整数后，我们便知道中数在那个范围内出现，以及这个范围内总共出现了多少个整数。  
　　如果中数所在范围出现的整数比较少，我们就可以对这个范围内的整数进行排序，找到中数。如果这个范围内出现的整数比较多，我们还可以采用同样的方法将此范围再次分成多个更小的范围（256M=2^28，所以最多需要3次就可以将此范围缩小到1，也就找到了中数）。  
19、时分秒针在一天之类重合多少次？（24小时）

2次

而时针和分针重合了22次。

20、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：{aaabbbccc}，{bbbddd}，{eeefff}，{ggg}，{dddhhh}要求将其中交集不为空的集合合并，要求合并完成后的集合之间无交集，例如上例应输出{aaabbbcccdddhhh}，{eeefff}，{ggg}。  
　　（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  
　　（3）请描述可能的改进。  
回答：  
　　集合使用hash\_set来表示，这样合并时间复杂度比较低。  
　　1、给每个集合编号为0，1，2，3...  
　　2、创建一个hash\_map，key为字符串，value为一个链表，链表节点为字符串所在集合的编号。遍历所有的集合，将字符串和对应的集合编号插入到hash\_map中去。  
　　3、创建一个长度等于集合个数的int数组，表示集合间的合并关系。例如，下标为5的元素值为3，表示将下标为5的集合合并到下标为3的集合中去。开始时将所有值都初始化为-1，表示集合间没有互相合并。在集合合并的过程中，我们将所有的字符串都合并到编号较小的集合中去。  
　  遍历第二步中生成的hash\_map，对于每个value中的链表，首先找到最小的集合编号（有些集合已经被合并过，需要顺着合并关系数组找到合并后的集合编号），然后将链表中所有编号的集合都合并到编号最小的集合中（通过更改合并关系数组）。  
　　4、现在合并关系数组中值为-1的集合即为最终的集合，它的元素来源于所有直接或间接指向它的集合。  
　　算法的复杂度为O(n)，其中n为所有集合中的元素个数。  
　　题目中的例子：  
　　0:{aaabbbccc}  
　　1:{bbbddd}  
　　2:{eeefff}  
　　3:{ggg}  
　　4:{dddhhh}  
　　生成的hash\_map，和处理完每个值后的合并关系数组分别为  
　　aaa:0。[-1,-1,-1,-1,-1]  
　　bbb:0,1。[-1,0,-1,-1,-1]  
　　ccc:0。[-1,0,-1,-1,-1]  
　　ddd:1,4。[-1,0,-1,-1,0]  
　　eee:2。[-1,0,-1,-1,0]  
　　fff:2。[-1,0,-1,-1,0]  
　　ggg:3。[-1,0,-1,-1,0]  
　　hhh:4。[-1,0,-1,-1,0]  
　　所以合并完后有三个集合，第0，1，4个集合合并到了一起，  
　21、平面内有11个点，由它们连成48条不同的直，由这些点可连成多少个三角形？  
解析：  
首先你要分析，平面中有11个点，如果这些点中任意三点都没有共线的，那么一共应该有C(11，2)=55，  可是，题目中说可以连接成48条直线，那么这11个点中必定有多点共线的情况。  55-48=7，从7来分析：  
假设有一组三个点共线，那么可以组成的直线在55的基础上应该减去C(3，2)-1=2     2\*3=6≠7，因此，可以断定不仅有三点共线的，也可能有四个点共线的可能。  
假设有一组四个点共线，那么可以组成的直线在55的基础上应该减去C(4，2)-1=5   
（备注，五个点共线的可能不存在，因为，C(5，2)-1=9>7，故不可能有五条直线共线。）  
  
因此，三点共线少2条，4点共线少5条，只有一个4点共线，一个3点共线才能满足条件，其余情况不能满足少了7条直线。  
那么，这11个点能组成的三角形的个数为，C(11，3)-C(3，3)-C(4，3)=165-1-4=160  （备注，三个点共线不能组成三角形）