（四）2011 笔经

1、对于一个内存地址是 32 位、内存页是 8KB 的系统。0X0005F123 这个地址的页号与页内偏移分别是多少。

页面大小是 8KB，那么页内偏移量是从 0x0000（0）~ 0x1FFF（2 的 13 次方 - 1）。0x5F123/8K=2E，余数是 1123；则页号是 47 页，页内偏移量应该是 0X00001123。

2、如果 X 大于 0 并小于 65536，用移位法计算 X 乘以 255 的值为：　　(X<<8)-X

X<<8-X 是不对的，因为移位运算符的优先级没有减号的优先级高，首先计算 8-X 为 0，X 左移 0 位还是 8。

3、一个包含 n 个节点的四叉树，每个节点都有四个指向孩子节点的指针，这 4n 个指针中有　　3n+1　　个空指针。

4、以下两个语句的区别是：第一个动态申请的空间里面的值是随机值，第二个进行了初始化，里面的值为 0

```cpp
int *p1 = new int[10];
int *p2 = new int[10]();
```

、

5、计算机在内存中存储数据时使用了大、小端模式，请分别写出 A=0X123456 在不同情况下的首字节是，大端模式：0X12 小端模式：0X56　　X86 结构的计算机使用　**小端**　　模式。

一般来说，大部分用户的操作系统（如 windows, FreeBsd,Linux）是小端模式的。少部分，如 MAC OS，是大端模式 的。

6、在游戏设计中，经常会根据不同的游戏状态调用不同的函数，我们可以通过函数指针来实现这一功能，请声明一个参数为 int *，返回值为 int 的函数指针：

int (*fun)(int *)

7、下面程序运行后的结果为：to test something

```cpp
char str[] = "glad to test something";
    char *p = str;
    p++;
    int *p1 = static_cast<int *>(p);
    p1++;
    p = static_cast<char *>(p1);
    printf("result is %s\n",p);
```

8、在一冒险游戏里，你见到一个宝箱，身上有 N 把钥匙，其中一把可以打开宝箱，假如没有任何提示，随机尝试，问：

（1）恰好第 K 次（1=<K<=N）打开宝箱的概率是多少。　(1-1/n)*(1-1/(n-1))*(1-1/(n-2))***(1/(n-k+1)) = 1/n

（2）平均需要尝试多少次。

这个就是求期望值　由于每次打开宝箱的概率都是 1/n，则期望值为：　1*(1/n)+2*(1/n)+3*(1/n)+.....+n*(1/n) = (n+1)/2

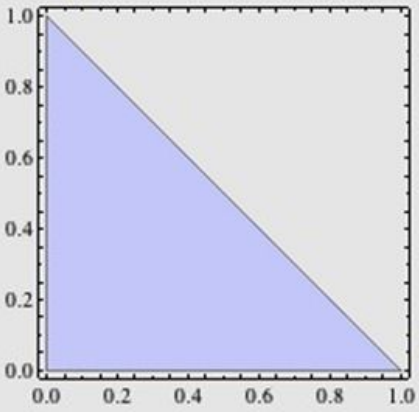
9、头文件中 ifndef / define / endif 是做什么用的？

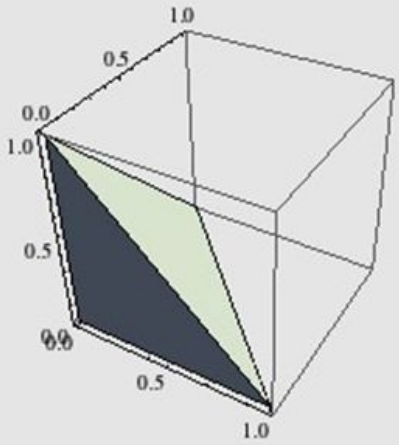
10、代码里有时可以看到 extern “C”，这语句是做什么用的？

11、平均要取多少个(0，1)中的随机数才能让和超过 1。

为了证明这一点，让我们先来看一个更简单的问题：任取两个 0 到 1 之间的实数，它们的和小于 1 的概率有多大？容易想到，满足 x+y<1 的点 (x, y) 占据了正方形 (0, 1)×(0, 1) 的一半面积，因此这两个实数之和小于 1 的概率就是 1/2 。类似地，三个数之和小于 的概率则是 1/6 ，它是平面 x+y+z=1 在单位立方体中截得的一个三棱锥。这个 1/6 可以利用截面与底面的相似比关系，通过简单的积分求得：

$$\int_0^1 (x^2)\cdot\frac{1}{2}\,dx = \frac{1}{6}$$





可以想到，四个 0 到 1 之间的随机数之和小于 1 的概率就等于四维立方体一角的"体积"，它的"底面"是一个体积为 1/6 的三维体，在第四维上对其进行积分便可得到其"体积"

$$\int_0^1 (x^3)\cdot\frac{1}{6}\,dx = \frac{1}{24}$$

依此类推， n 个随机数之和不超过 1 的概率就是 1/n! ，反过来 n 个数之和大于 1 的概率就是 1 - 1/n! ，因此加到第 n 个数才刚好超过 1 的概率就是

$$(1 - 1/n!) - (1 - 1/(n-1)!) = (n-1)/n!$$

因此，要想让和超过 1 ，需要累加的期望次数为

$$\sum_{n=2}^{\infty} n \cdot (n-1)/n! = \sum_{n=1}^{\infty} n/n! = e$$

12、在下列乘法算式中，每个字母代表 0~9 的一个数字，而且不同的字母代表不同的数字：

```
 ABCDEFGH
*                AJ
------------------
EJAHFDGKC
BDFHAJEC
------------------
CCCCCCCCC
```

请写出推导的过程。

本题唯一解为：A=2、B=4、C=6、D=9、E=1、F=3、G=5、H=8、J=7、K=0

13、输入格式：第一行输入 N（N<=100）表示流通的纸币面额数量；第二行 N 个纸币的具体表示的面额，从小到大排列，取值【1, 10^6】。

输出格式：输出一个整数，表示应该发行的纸币面额，这个整数是已经发行的所有纸币面额都无法表示的最小整数。（已经发行的每个纸币面额最多只能使用一次）

输入	输出
5 1 2 3 9 1	7

5
1 2 4 9 1
8
5
1 2 4 7 1
15

思路：这是一个典型的母函数问题，一般的典型母函数如 G(x) = (1+x+x^2+x^3+x^4+x^5+....)*(1+x^2+x^4+x^6+x^8+x^10+....)*(1+x^3+x^6+x^9+x^12....).....
这个题目中的每个纸币只能够使用0次或1次，在上面的那个一般的母函数的基础上修改一下就行了，就很简单了。。
具体代码如下：

```cpp
#include <iostream>
using namespace std;

const int lmax=10000;
int c1[lmax+1],c2[lmax+1];

int main(void)
{
    int m,n,i,j,k,a[110];
    //计算的方法还是模拟手动运算，一个括号一个括号的计算，从前往后
    while (cin>>m && m)
    {
        n=0;
        for(i = 0; i < m; i++)
        {
            scanf("%d",&a[i]);
            n += a[i];
        }
        n += 5;     //有可能无法表示的那个数比所有纸币面额的总和还要大
        for(i = 0; i <= n; i++)
        {
            c1[i] = 0;
            c2[i] = 0;
        }
        for(i = 0; i < 2*a[0]; i += a[0])        //母函数的表达式中第一个括号内的各项系数
            c1[i] = 1;
        //第一层循环是一共有 n 个小括号，而刚才已经算过一个了，所以是从2 到 n
        // i 就是代表的母函数中第几个大括号中的表达式
        for(i = 2; i <= m; i++)
        {
            for(j = 0; j <= n; j++)              //j 就是指的已经计算出的各项的系数
            {
```

```
33.                 for (k = 0; k < 2*a[i-1]; k += a[i-1])      //k 就是指将要计算的那个括号中
34.                 {
35.                     c2[j+k] += c1[j];        //合并同类项，他们的系数要加在一起，所以是加法
36.                 }
37.             }
38.             for(j = 0; j <= n; j++)    // 刷新一下数据，继续下一次计算，就是下一个括号里面的
39.             {
40.                 c1[j] = c2[j];
41.                 c2[j] = 0;
42.             }
43.         }
44.         for(i = 1; i <= n; i++)
45.         {
46.             if(c1[i] == 0)
47.             {
48.                 cout<<i<<endl;      //找出第一个无法表示的纸币面额
49.                 break;
50.             }
51.         }
52.     }
53.     return 0;
54. }
```

网易校园招聘笔试题 A 卷(研发类笔试题)

第一部分(必做):计算机科学基础

1.(单选)软件设计中模块划分应该遵循的准则是:

A.低内聚低耦合

B.高内聚低耦合

C.低内聚高耦合

D.高内聚高耦合

[cpp] view plaincopy

1. 答：B
2. 内聚指模块内部各成分之间相关程度的度量 强度性低到高分成 偶然内聚 ：关系松散没什么联系 逻辑内聚：几个逻辑上相关的功能被放在同一模块中，如一个模块读取各种不同类型外设的输入，逻辑内聚的模块各成分在功能上并无关系。时间内聚：一个模块完成的功能必须在同一时间内执行，这些功能只是因为时间因素关联在一起。通信内聚：如果一个模块的所有成分都操作同一数据集或生成同一数据集，则称为通信内聚。顺序内聚：如果一个模块的各个成分和同一个功能密切相关，而且一个成分的输出作为另一个成分的输入，则称为顺序内聚。功能内聚：模块的所有成分对于完成单一的功能都是必须的，则称为功能内聚。信息内聚：模块完成多个功能，各个功能都在同一数据结构上操作，每一项功能有一个唯一的入口点。这个模块将根据不同的要求，确定该模块执行哪一个功能。
3. 耦合指模块之间的关联程度。耦合由高到低分成：内容耦合：当一个模块直接修改或操作另一个模块的数据时，或一个模块不通过正常入口而转入另一个模块时，这样的耦合被称为内容耦合。公共耦合：两个或两个以上的模块共同引用一个全局数据项，这种耦合被称为公共耦合。外部耦合：一组模块都访问同一全局简单变量而不是同一全局数据结构。控制耦合：一个模块通过接口向一个模块传递控制信号，接受信号的的模块根据信号值进行适当的动作。标记耦合：一个模块通过接口向另一个模块传递一个控制信号，接受信号的模块根据信号值

而进行适当的动作，这种耦合被称为控制耦合。数据耦合：模块之间通过参数传递数据。非直接耦合：两个模块间没有直接关系，完全通过主模块的控制和调用实现。

2. (单选)最坏情况下时间复杂度不是 n(n-1)/2 的排序算法是:
A.快速排序
B.冒泡排序
C.直接插入排序
D.堆排序

[cpp] view plaincopy

1. 答：D
2. 快排最差是每次 partion 得到的位置在数组的两端时出现如排序有序数组。
3. 每次 partion 全部比较一次，每一趟比较后一个数位置确定，下次比较少一个数字少比较一次
4. （1+...N-1）
5. 冒泡每一趟通过相邻数字间交换实现把最大或最小数排到数组两端
6. 比较次数为（1+...N-1）
7. 直接插入最差情况是在逆序时如从小到大排序时每个数都比前面的所有数小
8. 比较次数(1+..N-1)
9. 堆排序最好最差情况一样 都为 nlogn

3. 哈希表中解决冲突的方法通常可以分为 open addressing 和 chaining 两类,请分别解释这两类冲突解决方法的大致实现原理

[cpp] view plaincopy

1. 答：第一个使用冲突算法在哈希表中在此寻找合适位置，
2. 分为线性探测再散列，存储地址 D 发生冲突，则放到存储地址（D＋1）%m；若又发生冲突则放到存储地址（D＋2）%m 二次探测，ND = (D+di)%m; di 取 1*1,-1*1,2*2,-2*2,.....,K*K,-K*K (K≤m/2)。拉链法个将所有关键字为同义词的结点链接在同一个单链表中

4. 简单的链表结构拥有很好的插入删除节点性能,但随机定位(获取链表第 n 个节点)操作性能不佳,请你设计一种改进型的链表结构优化随机定位操作的性能,给出设计思路及其改进后随机定位操作的时间复杂度

[cpp] view plaincopy

1. 使用 Node * List[MAX]按顺序存储链表节点地址，随机访问时间复杂度 O（1）；相对的删除增加节点时间变成 O(n);

5. 什么是 NP 问题?列举典型的 NP 问题(至少两个)?对于一个给定的问题你通常如何判断它是否为 NP 问题?

[cpp] view plaincopy

1. NP 问题是可以在多项式时间内被确定机(通常意义的计算机)解决的问题.NP(Non-Deterministic Polynomial, 非确定多项式)问题,是指可以在多项式时间内被非确定机(它可以猜,他总是能猜到最能满足你需要的那种选择,如果你让他解决 n 皇后问题,他只要猜 n 次就能完成----每次都是那么幸运)解决的问题。经典问题有：旅行商问题 TSP Travelling Salesman Problem、子集和问题、Hamilton 回路、最大团问题
2. 判断方法：可以将时间的时间复杂度与某个 NP 问题的时间复杂度作比较

6. 以下是一个 tree 的遍历算法, queue 是 FIFO 队列,请参考下面的 tree,选择正确的输出

[cpp] view plaincopy

```
queue.push(tree.root)
while(true){
    node=queue.pop();
    output(node.value);//输出节点对应数字
    if(null==node)
       break;
    for(child_node in node.children){
        queue.push(child_node);
    }
}
```

A.1234567
B. 1245367
C. 1376254
D. 1327654

[cpp] view plaincopy

```
答：A
按层次遍历输出
```

第二部分(选作): C/C++程序设计
1.有三个类 A B C 定义如下,请确定 sizeof(A) sizeof(B) sizeof(C)的大小顺序,并给出理由
[cpp] view plaincopy

```
struct A{
    A() {}8 字节，有虚函数所以又虚指针 占 4 字节 加上 8 字节变量
    ~A() {}
    int m1;
    int m2;
};
struct B：public A{//8 字节 char 因为 4 字节对齐占 4 个字节，static 不存储在类中
    B() {}
    ~B() {}
    int m1;
    char m2;
    static char m3;
};
struct C{
    C() {}
    virtual~C() {}//四字节对齐变量一共占 8 字节，有虚函数加 4 字节虚指针一共 12 字节。
    int m1;
    short m2;
};
```

答：

[cpp] view plaincopy

```
A：8 字节变量
B：8 字节 char 因为 4 字节对齐占 4 个字节，static 不存储在类中
C：四字节对齐变量一共占 8 字节，有虚函数加 4 字节虚指针一共 12 字节。
```

2.请用 C++实现以下 print 函数,打印链表 I 中的所有元素,每个元素单独成一行 voidprint(const std::list<int> &I)

[cpp] view plaincopy

```
void print(const std::list<int> &I){
    for(const std::list<int>::const_itorator it = I.begin():it != I.end(): it++)//访问常量必须使用常迭代器。
        cout<<*it<<endl;
}
```

3. 假设某 C 工程包含 a.c 和 b.c 两个文件,在 a.c 中定义了一个全局变量 foo,在 b.c 中想访问这一变量时该怎么做?

[cpp] view plaincopy

```
答：使用 extern
```

4. C++中的 new 操作符通常完成两个工作,分配内存及其调用相应的构造函数初始化

请问:

1) 如何让 new 操作符不分配内存,只调用构造函数?

2) 这样的用法有什么用?

答：参考 http://blog.csdn.net/aixiaolin/article/details/7367237

[cpp] view plaincopy

```
使用定位放置 new
#include <new> // 必须 #include 这个，才能使用 "placement new"
#include "Fred.h" // class Fred 的声明
void someCode()
{
char memory[sizeof(Fred)]; // Line #1
void* place = memory; // Line #2
Fred* f = new(place) Fred(); // Line #3 (详见以下的“危险”)
// The pointers f and place will be equal
// ...
}
作用为：对于需要反复创建并删除的对象，可以降低分配释放内存的性能消耗
```

5. 下面这段程序的输出是什么?为什么?

[cpp] view plaincopy

```
class A{
public:
    A(){p();}
    virtual void p(){print("A")}
    virtual ~A(){p();}
};
class B{
public:
    B(){p();}
    void p(){print("B")}
    ~B(){p();}
};
```

```
13. int main(int, char**){
14.     A* a=new B();
15.     delete a;
16. }
```

答：

[cpp] view plaincopy

```
1.  输出：ABBA
2.  原因：先构造父类 再构造子类 子类构造完成前 virtual 无效，析构虚函数会先析构子类
```

6. 什么是 C++ Traits?并举例说明

[cpp] view plaincopy

```
1.  答：特性萃取
2.  template <class T>
3.  class Demol{
4.      typedef  T Type;
5.  }
6.  template <class T>//偏特化
7.  class Demol<T *>{
8.      typedef  T Type;
9.  }
```

第三部分(选作): JAVA 程序设计

[cpp] view plaincopy

```
1.  1. (单选)以下 Java 程序运行的结构是:
2.  public class Tester{
3.  public static void main(String[] args){
4.    Integer var1=new Integer(1);
5.    Integer var2=var1;
6.    doSomething(var2);
7.    System.out.print(var1.intValue());
8.    System.out.print(var1==var2);
9.  }
10. public static void doSomething(Integer integer){
11. integer=new Integer(2);
12. }
13. }
14. A. 1true
15. B. 2true
16. C. 1false
17. D. 2false
18. 2. (单选)往 OuterClass 类的代码段中插入内部类声明, 哪一个是正确的:
19. public class OuterClass{
20. private float f=1.0f;
21. //插入代码到这里
22. }
23. A.
24. class InnerClass{
25. public static float func(){return f;}
26. }
```

```
B.
abstract class InnerClass{
public abstract float func(){}
}
C.
static class InnerClass{
protected static float func(){return f;}
}
D.
public class InnerClass{
static static float func(){return f;}
}
```

3. Java 中的 interface 有什么作用? 举例说明哪些情况适合用 interface, 哪些情况下适合用抽象类.

4. Java 多线程有哪几种实现方式? Java 中的类如何保证线程安全? 请说明 ThreadLocal 的用法和适用场景

5. 线程安全的 Map 在 JDK 1.5 及其更高版本环境 有哪几种方法可以实现?

6.

1) 简述 Java ClassLoader 的模型, 说明其层次关系及其类加载的主要流程即可.

2) TypeA.class 位于 classpath 下, /absolute_path/TypeA.class 为其在文件系统中的绝对路径, 且类文件小于 1k, MyClassLoader 为一个自定义的类加载器, 下面的这段类加载程序是否正确, 如果有错请指出哪一行有错, 简述理由

```
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
public class Tester{
public static void main(String[] args){
   MyClassLoader cl1=new MyClassLoader();
   try{
    File f=new File("/absolute_path/TypeA.class");
    byte[] b=new byte[1024];
    InputStream is=new FileInputStream(f);
    int I=is.read(b);
    Class c=cl1.defineMyClass(null,b,0,1);
    TypeA a=(TypeA)c.newInstance();
   }catch(Exception e){
    e.printStacktrace();
   }
}
}
```

第四部分(选作): Linux 应用与开发

1. 写出完成以下功能的 Linux 命令:

1) 在当前目录及其子目录所有的.cpp 文件中查找字符串"example",不区分大小写;

2) 使用 sed 命令,将文件 xyz 中的单词 AAA 全部替换为 BBB;

3) 用一条命令创建 aabb cc 三个子目录

4) mount cdrom.iso 至/dev/cdrom 目录

5) 设置 ulimit 使得程序在 Segment fault 等严重错误时可以产生 coredump;

2. 设 umask 为 002,则新建立的文件的权限是什么?

A. -rw-rwr--

B. rwxrwx-w-

C. -------w-

D. rwxrwxr-x
3. 用户 HOME 目录下的.bashrc 和.bash_profile 文件的功能有什么区别?
4. 写出完成以下功能的 gdb 命令(可以使用命令简写形式):
1) 使用 gdb 调试程序 foo,使用 coredump 文件 core.12023;
2) 查看线程信息
3) 查看调用堆栈
4) 在类 ClassFoo 的函数 foo 上设置一个断点
5) 设置一个断点,当表达式 expr 的值被改变时触发
5.
1) 例举 Linux 下多线程编程常用的 pthread 库提供的函数名并给出简要说明(至少给出 5 个)
2) pthread 库提供哪两种线程同步机制,列出主要 API
3) 使用 pthread 库的多线程程序编译时需要加什么连接参数?

第五部分(选作): Windows 开发
1. DC(设备上下文)有哪几类?区别在哪里?
[cpp] view plaincopy

1. CpaintDC 在窗口的成员函数 OnPaint 中使用的一种设备上下文，在构造过程中自动调用 BeginPaint 析构时自动调用 EndPaint
2. CClientDC 代表客户区域的设备上下文
3. CWindowDC 代表整个窗口的设备上下文
4. CMetaFileDC 代表 Windows 图元文件的设备上下文

2. 碰撞检测是游戏中经常要用到的基本技术对于二维情况,请回答以下问题:
1). 如何判断一个点在一个多边形内
答：四边形上下左右边界判断
2). 如何判断两个多边形相交
答：判断 A 四边形每条边与 B 多边形的每边相交情况，如果有相交则多边形相交
3). 如何判断两个点集所形成的完全图所围的区域是否相交
答：求凸包然后然后问题变成判断多边形是否相交

3.PostMessage SendMessage 和 PostThreadMessage 的区别是什么
[cpp] view plaincopy

1. 答：post 和 send 把消息送进指定窗口创建的线程的消息队列，send 会等待消息处理后继续执行，post 直接处理下一条语句，postThread 直接把消息传递给对应线程的消息队列

4. 什么叫 Alpha 混合?当前流行的图片格式中哪些支持 alpha 通道?LayeredWindow 和普通 Window 有什么区别?
[cpp] view plaincopy

1. 答：
2. （1）将要绘制的物体颜色与颜色缓冲区中存在的颜色相混合，从而绘制出具有半透明效果的物体。
3. 假设一种不透明东西的颜色是 A，另一种透明的东西的颜色是 B，那么透过 B 去看 A，看上去的颜色 C 就是 B 和 A 的混合颜色，可以用这个式子来近似，设 B 物体的透明度为 alpha(取值为 0-1，0 为完全透明，1 为完全不透明)
4. R(C)=alpha*R(B)+(1-alpha)*R(A)
5. G(C)=alpha*G(B)+(1-alpha)*G(A)
6. B(C)=alpha*B(B)+(1-alpha)*B(A)
7. R(x)、G(x)、B(x)分别指颜色 x 的 RGB 分量。
8. （2）PNG TGA
9. （3）Layered Window 可以实现像素级的透明度调整而普通 window 只能整体调整透明度

5. 如果要实现一个多线程(非 MFC)程序,选择多线程 CRT, 创建线程的时候应该用 CreateThread 还是_beginthreadex(), 为什么?
为何要用 _beginthreadex()而非 CreateThread?

答：参考 http://blog.csdn.net/shu_nt/article/details/7543528
[cpp] view plaincopy

1. 传统的 CRT 不支持多线程，为了能正常使用，_beginthreadex()使用了系统的 TlsGetValue 函数来获取对应线程的 tiddata 内存块地址，使 tiddata 与线程关联互不影响。
2. 如果要作多线程(非 MFC)程序，在主线程以外的任何线程内
3. - 使用 malloc(),free(),new
4. - 调用 stdio.h 或 io.h,包括 fopen(),open(),getchar(),write(),printf(),errno
5. - 使用浮点变量和浮点运算函数
6. - 调用那些使用静态缓冲区的函数如: asctime(),strtok(),rand()等。
7. 你就应该使用多线程的 CRT 并配合_beginthreadex(该函数只存在于多线程 CRT), 其他情况下你可以使用单线程的 CRT 并配合 CreateThread。
8. 因为对产生的线程而言，_beginthreadex 比之 CreateThread 会为上述操作多做额外的簿记工作，比如帮助 strtok()为每个线程准备一份缓冲区。
9. 然而多线程程序极少情况不使用上述那些函数(比如内存分配或者 io)，所以与其每次都要思考是要使用 _beginthreadex 还是 CreateThread,不如就一棍子敲定_beginthreadex。
10. 当然你也许会借助 win32 来处理内存分配和 Io，这时候你确实可以以单线程 crt 配合 CreateThread，因为 io 的重任已经从 crt 转交 给了 win32。这时通常你应该使用 HeapAlloc，HeapFree 来处理内存分配，用 CreateFile 或者 GetStdHandle 来处理 Io。
11. 还有一点比较重要的是_beginthreadex 传回的虽然是个 unsigned long,其实是个线程 Handle(事实上 _beginthreadex 在内部就是调用了 CreateThread)，所以你应该用 CloseHandle 来结束他。千万不要使用 ExitThread()来退出_beginthreadex 创建的线程，那样会丧失释放簿记数据的机会， 应该使用_endthreadex.

第六部分(选作): 数据库开发
1. 基于哈希的索引和基于树的索引有什么区别?
2. User 表用于记录用户相关信息, Photo 表用于记录用户的照片信息, 两个表的定义如下:

```
CREATE TABLE User( --用户信息表
UserId bigint,  --用户唯一 id
Account varchar(30)  --用户唯一帐号
);
CREATE TABLE Photo(  --照片信息表
PhotoId bigint, --照片唯一 id
UserId bigint,  --照片所属用户 id
AccessCount int,  --访问次数
Size bigint  --照片文件实际大小
)
```

1) 请给出 SQL 打印帐号为"dragon"的用户访问次数最多的 5 张照片的 id;
2) 给出 SQL 打印拥有总的照片文件大小(total_size)最多的前 10 名用户的 id, 并根据 total_size 降序排列
3) 为优化上面两个查询, 需要在 User 和 Photo 表上建立什么样的索引?
4) 简述索引对数据库性能的影响?
3. 什么是两阶段提交协议?
4. 数据库事务基本概念:
1) 什么是事务的 ACID 性质?
2) SQL 标准中定义的事务隔离级别有哪四个?
3) 数据库中最常用的是哪两种并发控制协议?
4) 列举你所知的数据库管理系统中采用的并发控制协议

5. 数据库中有表 User(id, name, age):
表中数据可能会是以下形式:
id name age
001 张三 56
002 李四 25
003 王五 56
004 赵六 21
005 钱七 39
006 孙八 56
.....
由于人员年龄有可能相等, 请写出 SQL 语句, 用于查询 age 最大的人员中, id 最小的一个记录
6. 并发访问数据库时常使用连接池, 请问使用连接池的好处是什么? 对于有多台应用服务器并发访问一台中心数据库的情况, 数据库访问往往成为系统瓶颈, 请问在应用服务器上设计和使用连接池时该注意哪些问题, 以保证系统的可靠性正确性和整体性能. 假设每台应用服务器都执行相同的任务并且负载均衡.

第七部分(选作): Web 开发
1. 以下哪一条 Javascript 语句会产生运行错误:
A. var obj=( );
B. var obj=[ ];
C. var obj={ };
D. var obj=/ /;
2. 如下页面代码(示例代码 DOCTYPE 为 Strict)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="zh"lang="zh">
<head>
<title>测试</title>
<meta http-equiv="content-type" content="text/html;charset=gbk" />
<meta http-equiv="content-style-type" content="text/cee"/>
<meta http-equiv="content-script-type"content="text/javascript" />
<script type="text/css">
*{margin:0; padding:0}
  html{width:100%;height:100%;
   overflow:scroll;overflow-x:auto;
   text-align:center;border:0}
  .test{height:200px}
</script>
</head>
<body>
<div class="text"> </div>
</body>
</html>
```

假设 a.jpg 图片的规格是 200pxX100px, 请给出当前背景图片距 div.a 顶部距离的计算方式和结果(css)
3. HTTP 协议相关知识
A) 常见的 HTTPRequest 头字段有哪些?
B) web 服务器如何区分访问者是普通浏览用户还是搜索引擎的 Spider?
C) cookie 按生命周期分类分为哪两类? 其生命周期分别是多长?向浏览器设置 cookie 时 cookie 有哪些属性可以设置, 分别起到什么作用?
D) HTTP 协议中 Keep-Alive 是什么意思? 使用 Keep-Alive 有何好处, 对服务器会有什么不利的影响? 对于不利的影响有什么解决方案

4. 简述你最常用的 Web 服务器的一种或者几种, 并说明如何在 Web 服务器和应用服务器之间建立反向代理
5. 简述你所了解的 MVC 各层次的常用开发框架, 说明其特点
6. 简述 Web 应用环境下远程调用的几种方式, 并且从性能异构性等方面比较其优劣

第八部分(选作): Flash 开发
1.flash 和 js 如何交互?
2. flash 中的事件处理分哪几个过程 Event 对象的 target 和 currentTarget 有什么区别?

第九部分(选作): 软件测试
1. 请描述你对测试的了解, 内容可以涉及测试流程, 测试类型, 测试方法, 测试工具等
2. 如果有一天你早上上班,发现不能上网了, 你会用什么步骤找出问题所在?
3. Web 应用中实现了好友功能,用户可以给别人发"加为好友"的请求, 发了请求后可以取消请求, 对方收到请求后, 可以选择接受或者拒绝. 互为好友的两个人, 每个人都可以单方面删除对方, 请设想尽可能多的路径对此功能设计测试用例, 每个用例包括测试步骤和预期结果
4. 公司开发了一个 web 聊天工具, 用于网络用户之间的聊天, 一个人同时可以和多个人聊天, 功能类似于 MSN 等等 IM 工具
要求该系统能承受 1 万个在线用户, 平均每个用户会和 3 个人同时聊天, 在网络条件正常的情况下, 要求用户收到消息的延迟时间不超过 1 分钟. 现在需要对系统进行性能测试, 验证系统是否达到预定要求, 请你写一个性能测试方案. 提示如下:
1) 性能测试的过程一般都是模拟大量客户端操作, 同时监控服务器的性能和客户端相应, 根据服务器的性能指标和客户端响应状况进行分析和判断
2) 系统的性能问题可以从两个角度考虑, 一个是服务器问题,设计得不好的程序, 在大负载或者长时间运行情况下, 服务器会 down 机; 另一个是客户端问题, 在负载大的时候, 客户端响应会变慢
3) 在答题中,可以不涉及性能测试工具, 监控工具等细节, 把你的测试思路说清楚就可以
5. 自动功能测试中会将测试用例组织成测试集合来统一运行, 测试集合 suite 按功能分类可以有若干个模块 module, 每个模块 module 下包含若干个测试用例 test. 现测试集合已经运行完毕, 但是需要在测试报告中统计各个模块的用例失败率, 将失败率超过 20%的模块名与其失败率记录下来报警, 请编写实现上述功能的 getTestReport 函数. 可使用 Java 或 C++等您熟悉的编程语言, 提供的接口及方法如下:
测试集合接口 Isuite:
Collection<ITest>getTests() //得到测试集合下的所有测试用例 test
测试用例接口 Itest:
String getModule() //得到该用例对应的模块名称 module
int getResult() //得到该用例的执行结果:0 失败 1 成功
报警函数:
void alertMessage(String message)
public static void getTestReport(ISuite suite){
  //你的实现写在这里
}
网易 C++笔试题 1. #i nclude 和#i nclude “filename.h” 有什么区别?
答：对于#i nclude ，编译器从标准库路径开始搜索 filename.h
对于#i nclude “filename.h”，编译器从用户的工作路径开始搜索 filename.h
网易 C++笔试题 2. 在 C++ 程序中调用被 C 编译器编译后的函数，为什么要加 extern “C”?
答：C++语言支持函数重载，C 语言不支持函数重载。函数被 C++编译后在库中的名字与 C 语言的不同。假设某个函数的原型为： void foo(int x, int y);
该函数被 C 编译器编译后在库中的名字为_foo ， 而 C++ 编译器则会产生像_foo_int_int 之类的名字。
C++提供了 C 连接交换指定符号 extern“C”来解决名字匹配问题。
网易 C++笔试题 3. 一个类有基类、内部有一个其他类的成员对象，构造函数的执行顺序是怎样的?
答：先执行基类的(如果基类当中有虚基类，要先执行虚基类的，其他基类则按照声明派生类时的顺序依次执行)，再执行成员对象的，最后执行自己的。
网易 C++笔试题 4. New delete 与 malloc free 的区别

答案：用 malloc 函数不能初始化对象，new 会调用对象的构造函数。Delete 会调用对象的 destructor，而 free 不会调用对象的 destructor.

网易 C++笔试题 5. Struct 和 class 的区别

答案：struct 中成员变量和成员函数默认访问权限是 public,class 是 private

网易 C++笔试题 6.请问下面程序有什么错误?

```
int a[60][250][1000],i,j,k;
for(k=0;k<=1000;k++)
for(j=0;j<250;j++)
for(i=0;i<60;i++)
a[i][j][k]=0;
```

答案：把循环语句内外换一下

网易 C++笔试题 7. 请写出下列代码的输出内容

```
#include
main()
{
int a,b,c,d;
a=10;
b=a++;
c=++a;
d=10*a++;
printf("b，c，d：%d，%d，%d"，b，c，d);
return 0;
}
```

答：10，12，120

网易 C++笔试题 8. 写出 BOOL,int,float,指针类型的变量 a 与零的比较语句。

答案： BOOL : if (!a)

```
int : if ( a == 0)
float : const EXPRESSION EXP = 0.000001
if ( a < EXP && a >-EXP)
pointer : if ( a != NULL)
```

网易 C++笔试题 9.已知 strcpy 函数的原型是：

```
char *strcpy(char *strDest, const char *strSrc);
```

其中 strDest 是目的字符串，strSrc 是源字符串。不调用 C++/C 的字符串库函数，请编写函数 strcpy

答案：

```
char *strcpy(char *strDest, const char *strSrc)
{
if ( strDest == NULL || strSrc == NULL)
return NULL ;
if ( strDest == strSrc)
return strDest ;
char *tempptr = strDest ;
while( (*strDest++ = *strSrc++) != ‘’)
;
return tempptr ;
}
```

网易 C++笔试题 10.写一个函数找出一个整数数组中，第二大的数

答案：

```
const int MINNUMBER = -32767 ;
int find_sec_max( int data[] , int count) //类似于 1 4 4 4 这样的序列将认为 1 是第二大数
```

```
{
int maxnumber = data[0] ;
int sec_max = MINNUMBER ;
for ( int i = 1 ; i < count ; i++)
{
if ( data[i] > maxnumber )
{
sec_max = maxnumber ;
maxnumber = data[i] ;
}
else
{
if ( data[i] > sec_max )
sec_max = data[i] ;
}
}
return sec_max ;
}
```

1.哈希表的实现方式有哪几种，实现一种 hash_insert

2.写一个程序，打印出以下的序列。

(a),(b),(c),(d),(e).....(z)
(a,b),(a,c),(a,d),(a,e).....(a,z),(b,c),(b,d).....(b,z),(c,d).....(y,z)
(a,b,c),(a,b,d)....(a,b,z),(a,c,d)....(x,y,z)
....
(a,b,c,d,.....x,y,z)

3.给出示例代码，如何限制一个类只在堆上分配和栈上分配

4.大概是下面这个样子吧，但愿我没把函数调用的地方记错。。。

```
#include <iostream>
using namespace std;

class A
{
public:
    A(int j):i(j)
    {
        fun1();
    }
    ~A()
    {
    }

    virtual void fun2()
    {
        i++;
    }
    void fun1()
```

```
    {
        i *= 10;
    }

    int i;
};

class B:public A
{
public:
    B(int j):A(j)
    {
        fun2();
    }
    ~B()
    {
    }

    void fun2()
    {
        i += 2;
    }
    void fun1()
    {
        i *= 100;
    }
};

void main()
{
    A* p = new B(1);
    cout<<p->i<<endl;
    delete p;
};
```

5. 我觉得这个题目怎么这么诡异，我觉得编译通过不过，A 和 B 没有继承关系，我没记错。。。

```
#include <iostream>
using namespace std;

class A
{
public:
    A() { }
 ~A() { }
virtual void fun();
};
```

```
void A::fun()
{
 ...
}

class B
{
public:
    B() { }
 ~B() { }
    virtual void fun();
};

void B::fun()
{
  ...
}
void main()
{
    A* p = new B;
    t1->fun(); // 这个 t1 是怎么回事，应该是 p 吧，这个应该是出试卷的时候打字错误
};
```

6. 改错题，为什么弄这么多 static 变量，我就纳闷了，考 static 知识点一个也就够了，弄这么多个

```
#include <iostream>
using namespace std;

class A
{
public:
    A();
    ~A();

    int i = 0;
    static int j = 0;
    const int k = 0;
    const static char *p = "Hello world";
    static void fun();
};

A::A()
{

}

A::~A()
{
```

```
}

static void fun()
{

}
```

试题第一题是七巧板拼图，他拼出三把游戏中常见的宝剑图形，让我们再拼出三种图形，然后说一下自己认为拼七巧板的技巧。刚开始拿到这个题，不知道怎么下手，先跳过（最后这道题是在交试卷前 5 分钟匆忙做完的）。考完才感觉这道题是相当简单，当时刚考试考有点紧张，竟不知所措！（心理素质有待加强！）

试题第二题是给出四个图形，然后让你找出不同于其他三个的那个图形（这种题好像在公务员考试中常见），给出的四个图形分别是圆、半圆、两个同心圆和一个三角形。然后给出那个不同其他三个的原因。这个题还有后面的那个推理题，自己把原因分析的有点太细，浪费了不少时间。以后再笔试要注意点，该言简意赅的就言简意赅，别低估阅卷人的智商。先把基本问题解决了，再去考虑如何去优化。

第三题是：一个 100 米长的部队行军，一个传令兵从队伍尾到队伍头，然后再从队伍头返回队伍尾，这时队伍正好走了 100 米，整个过程队伍和传令兵的速度不变，那么传令兵走了多少米？这个题说实话有点像小学数学题，我为了节省时间，采用了参照物和相对位移的方法，当自我感觉良好的时候，出来和大部队的结果不同！汗！

第四题是一个推理题，由于题干很长，我只说大体意思。题的大意是：有五个同学被蒙上眼睛，然后每个人戴一顶帽子，帽子分两种颜色，黑和白，然后五个人都猜除了自己以外，其他人所戴帽子是黑色和白色的个数，……当猜完第一轮，老师说戴白色帽子的人都猜对了，戴黑色帽子的人猜错了。然后再让大家猜第二轮，第二轮大家都猜出每个同学所戴的帽子的颜色。最后问每个人所戴帽子的颜色，并详细写出推理过程。这个题不难，只是写推理过程有点费时间。

第五题是技术题了，是根据要求画出流程图，并写出程序。题干很长，主要是判断的地方很多。省略。值得一提的是，这个题提供了一些函数接口让你去用。

第六题是概率统计题，邮件被分为垃圾邮件（A）和非垃圾邮件（B），然后单词 w 在 A 出现的概率 Pa=在 A 中出现 w 的次数/总单词数，Pb=在 B 中出现 w 次数/单词总数。求（1）出现单词 w 的邮件是垃圾邮件的概率。（2）……（第二问没有看，跳过了）

第七题是英文题，跳过了，因为英文就看的乱七八糟的，更不用说去分析了。（英语学习得加强，北电也有英文题的）。

紧接着后面三个题是游戏相关的，一个是让你设计一个角色的数据结构，要包括一些属性（生命值、攻击力、防御力、魔法），然后下面每一问都让你根据问题来修改数据结构，来完善这个角色的定义。第二个是一个打怪的题，给你一个图，上面有一些怪，坐标固定，打怪消耗的能量和离怪物的距离成正比，然后让你找出最合适的位置（沿 Y 轴东西方向移动，打怪是南北方向），打掉所有的怪消耗能量最小。第三题是根据不同付费用户，如：按具体时间付费用户、包月用户、特权用户（免费），设计用户信息数据结构，并实现付费函数。（我想这个题应该是考的虚函数，我是那样写的，不知道对不。）

两个附加题，一数理相关，一英文题，看了几眼，印象不深，没有记住。（实在是没有时间去做了，为自己的答题速度担心！）

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 http://www.cppentry.com 后，你看到了首页，那么这一切是如何发生的呢？浏览器做了什么？服务器又是怎么返回的？

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，ISP DNS 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。

Traits 技术可以用来获得一个 类型 的相关信息的。 首先假如有以下一个泛型的迭代器类，其中类型参数 T 为迭代器所指向的类型：

```
template <typename T>
class myIterator
{
...
};
```
当我们使用 myIterator 时，怎样才能获知它所指向的元素的类型呢？我们可以为这个类加入一个内嵌类型，像这样：
```
template <typename T>
class myIterator
{
	typedef T value_type;
...
};
```
这样当我们使用 myIterator 类型时，可以通过 myIterator::value_type 来获得相应的 myIterator 所指向的类型。
现在我们来设计一个算法，使用这个信息。
```
template <typename T>
typename myIterator<T>::value_type Foo(myIterator<T> i)
{
...
}
```
这里我们定义了一个函数 Foo，它的返回为为 参数 i 所指向的类型，也就是 T，那么我们为什么还要兴师动众的使用那个 value_type 呢？ 那是因为，当我们希望修改 Foo 函数，使它能够适应所有类型的迭代器时，我们可以这样写：
```
template <typename I> //这里的 I 可以是任意类型的迭代器
typename I::value_type Foo(I i)
{
...
}
```
现在，任意定义了 value_type 内嵌类型的迭代器都可以做为 Foo 的参数了，并且 Foo 的返回值的类型将与相应迭代器所指的元素的类型一致。至此一切问题似乎都已解决，我们并没有使用任何特殊的技术。然而当考虑到以下情况时，新的问题便显现出来了：
原生指针也完全可以做为迭代器来使用，然而我们显然没有办法为原生指针添加一个 value_type 的内嵌类型，如此一来我们的 Foo()函数就不能适用原生指针了，这不能不说是一大缺憾。那么有什么办法可以解决这个问题呢？ 此时便是我们的主角：类型信息榨取机 Traits 登场的时候了
....drum roll.....
我们可以不直接使用 myIterator 的 value_type，而是通过另一个类来把这个信息提取出来：
```
template <typename T>
class Traits
{
	typedef typename T::value_type value_type;
};
```
这样，我们可以通过 Traits<myIterator>::value_type 来获得 myIterator 的 value_type，于是我们把 Foo 函数改写成：
```
template <typename I> //这里的 I 可以是任意类型的迭代器
typename Traits<I>::value_type Foo(I i)
{
...
}
```
然而，即使这样，那个原生指针的问题仍然没有解决，因为 Trait 类一样没办法获得原生指针的相关信息。于是我们祭出 C++的又一件利器--偏特化(partial specialization)：
```
template <typename T>
class Traits<T*> //注意 这里针对原生指针进行了偏特化
```

```
{
    typedef typename T value_type;
};
```

通过上面这个 Traits 的偏特化版本，我们陈述了这样一个事实：一个 T* 类型的指针所指向的元素的类型为 T。如此一来，我们的 Foo 函数就完全可以适用于原生指针了。比如：

```
int * p;
....
int i = Foo(p);
```

Traits 会自动推导出 p 所指元素的类型为 int，从而 Foo 正确返回。

常见的问题有以下几类：

问题一：有 101 个数，为[1，100]之间的数，其中一个数是重复的，如何寻找这个重复的数，其时间复杂度和空间复杂度是多少？

方法：利用和

sum1=1+2+3+.....99;

sum2=a[0]+a[1]+.....a[99];

sum2-sum1=重复的那个值

```
int OnlyOneRepeate(int *iArray,int length)
{
    int i,sum = 0,sumMax = 0;
    for(i = 0;i < length; i++)
    {
        sum += i;
        sumMax += iArray[i];
    }
    return sumMax - sum;
}
```

另外的方法：我们可以建立一个长度为 length 大小的数组，并且初始化为 0：

```
int *nCount = new int [length];
nCount[ ] = {0};
for(int i = 0;i < length; i++)
{
     nCount [iArray[ i ] ] ++;
     if(2 == nCount[ iArray[ i ] ])
          return iArray[i];
}
```

此算法就是申请了一个大小为:length*4 字节的空间，在一定程度上空间复杂度提高了；但是，在时间复杂度上有所降低。

问题二：有大量的数据，其中只有一个数字出现了一次，其他的数字至少出现了两次，如何找出这个只出现一次的数字，要求用最小时间复杂度和空间复杂度。

方法：利用 a^b^b=a 的原理

直接 a[0]^a[1]^.....a[99] = 只出现一次的那个值.

问题三：数组中有一个数字出现的次数超过了数组长度的一半，找出这个数字。

数组中有个数字出现的次数超过了数组长度的一半。也就是说，有个数字出现的次数比其他所有数字出现次数的和还要多。因此我们可以考虑在遍历数组的时候保存两个值：一个是数组中的一个数字，一个是次数。当我们遍历到下一个数字的时候，如果下一个数字和我们之前保存的数字相同，则次数加 1。如果下一个数字和我们之前保存的数字不同，则次数减 1。如果次数为零，我们需要保存下一个数字，并把次数设为 1。由于我们要找的数字出现的次数比其他所有数字出现的次数之和还要多，那么要找的数字肯定是最后一次把次数设为 1 时对应的数字。

基于这个思路，我们不难写出如下代码：

```
bool g_bInputInvalid = false;
////////////////////////////////////////////////////////////////////////////
// Input: an array with "length" numbers. A number in the array
// appear more than "length / 2 + 1" times.
// Output: If the input is valid, return the number appearing more than
// "length / 2 + 1" times. Otherwise, return 0 and set flag g_bInputInvalid
// to be true.
////////////////////////////////////////////////////////////////////////////
int MoreThanHalfNum(int* numbers, unsigned int length)
{
    if(numbers == NULL && length == 0)
    {
        g_bInputInvalid = true;
        return 0;
    }
    g_bInputInvalid = false;
    int result = numbers[0];
    int times = 1;
    for(int i = 1; i < length; ++i)
    {
        if(times == 0)
        {
            result = numbers[i];
            times = 1;
        }
        else if(numbers[i] == result)
            times++;
        else
            times--;
    }
    // verify whether the input is valid
    times = 0;
    for(int i = 0; i < length; ++i)
```

```
    {
        if(numbers[i] == result)
            times++;
    }
    if(times * 2 <= length)
    {
        g_bInputInvalid = true;
        result = 0;
    }
    return result;
}
```

在上述代码中，有两点值得讨论：

（1）我们需要考虑当输入的数组或者长度无效时，如何告诉函数的调用者输入无效。关于处理无效输入的几种常用方法；

（2）本算法的前提是输入的数组中的确包含一个出现次数超过数组长度一半的数字。如果数组中并不包含这么一个数字，那么输入也是无效的。因此在函数结束前我还加了一段代码来验证输入是不是有效的。

何谓海量数据处理？

所谓海量数据处理，其实很简单，海量，海量，何谓海量，就是数据量太大，所以导致要么是无法在较短时间内迅速解决，要么是数据太大，导致无法一次性装入内存。

那解决办法呢?针对时间，我们可以采用巧妙的算法搭配合适的数据结构，如 Bloom filter/Hash/bit-map/堆/数据库或倒排索引/trie/，针对空间，无非就一个办法：大而化小：分而治之/hash 映射，你不是说规模太大嘛，那简单啊，就把规模大化为规模小的，各个击破不就完了嘛。

至于所谓的单机及集群问题，通俗点来讲，单机就是处理装载数据的机器有限(只要考虑 cpu，内存，硬盘的数据交互)，而集群，机器有多辆，适合分布式处理，并行计算(更多考虑节点和节点间的数据交互)。

再者，通过本 blog 内的有关海量数据处理的文章，我们已经大致知道，处理海量数据问题，无非就是：

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。

下面，本文第一部分、从 set/map 谈到 hashtable/hash_map/hash_set，简要介绍下 set/map/multiset/multimap，及 hash_set/hash_map/hash_multiset/hash_multimap 之区别(万丈高楼平地起，基础最重要)，而本文第二部分，则针对上述那 6 种方法模式结合对应的海量数据处理面试题分别具体阐述

第一部分、从 set/map 谈到 hashtable/hash_map/hash_set

稍后本文第二部分中将多次提到 hash_map/hash_set，下面稍稍介绍下这些容器，以作为基础准备。一般来说，STL 容器分两种，

- 序列式容器(vector/list/deque/stack/queue/heap)，
- 关联式容器。关联式容器又分为 set(集合)和 map(映射表)两大类，以及这两大类的衍生体 multiset(多键集合)和 multimap(多键映射表)，这些容器均以 RB-tree 完成。此外，还有第 3 类关联式容器，如 hashtable(散列表)，以及以 hashtable 为底层机制完成的 hash_set(散列集合)/hash_map(散列映射表)/hash_multiset(散列多键集合)/hash_multimap(散列多键映射表)。也就是说，set/map/multiset/multimap 都内含一个 RB-tree，而 hash_set/hash_map/hash_multiset/hash_multimap 都内含一个 hashtable。

所谓关联式容器，类似关联式数据库，每笔数据或每个元素都有一个键值(key)和一个实值(value)，即所谓的 Key-Value(键-值对)。当元素被插入到关联式容器中时，容器内部结构(RB-tree/hashtable)便依照其键值大小，以某种特定规则将这个元素放置于适当位置。

包括在非关联式数据库中，比如，在 MongoDB 内，文档(document)是最基本的数据组织形式，每个文档也是以 Key-Value(键-值对)的方式组织起来。一个文档可以有多个 Key-Value 组合，每个 Value 可以是不同的类型，比如 String、Integer、List 等等。

```
{ "name" : "July",
  "sex" : "male",
   "age" : 23 }
```

set/map/multiset/multimap

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

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。

所以，综上，说白了，什么样的结构决定其什么样的性质，因为 set/map/multiset/multimap 都是基于 RB-tree 之上，所以有自动排序功能，而 hash_set/hash_map/hash_multiset/hash_multimap 都是基于 hashtable 之上，所以不含有自动排序功能，至于加个前缀 multi_无非就是允许键值重复而已。

此外，

- 关于什么 hash，请看 blog 内此篇文章：http://blog.csdn.net/v_JULY_v/article/details/6256463；
- 关于红黑树，请参看 blog 内系列文章：http://blog.csdn.net/v_july_v/article/category/774945，
- 关于 hash_map 的具体应用：http://blog.csdn.net/sdhongjun/article/details/4517325，关于 hash_set：http://blog.csdn.net/morewindows/article/details/7330323。

OK，接下来，请看本文第二部分、处理海量数据问题之六把密匙。

第二部分、处理海量数据问题之六把密匙

密匙一、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。

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。

或者如下阐述（雪域之鹰）：

算法思想：分而治之+Hash

1. IP 地址最多有 2^32=4G 种取值情况，所以不能完全加载到内存中处理；
2. 可以考虑采用“分而治之”的思想，按照 IP 地址的 Hash(IP)%1024 值，把海量 IP 日志分别存储到 1024 个小文件中。这样，每个小文件最多包含 4MB 个 IP 地址；
3. 对于每一个小文件，可以构建一个 IP 为 key，出现次数为 value 的 Hash_map，同时记录当前出现次数最多的那个 IP 地址；
4. 可以得到 1024 个小文件中的出现次数最多的 IP，再依据常规的排序算法得到总体上出现次数最多的 IP；

分析：有的网友提出以下疑问：我感觉这个值不应该是要求的那个。因为可能某一个 ip 在某一个小文件中可能出现频率很高，但是在其他小文件中可能没出现几次，即**分布不均**，但因为某一个小文件中特别多而被选出来了；而另一个 ip 可能在每个小文件中都不是出现最多的，但是它在每个文件中都出现很多次，即**分布均匀**，因此非常有可能它就是总的出现次数最多的，但是因为在每个小文件中出现的次数都不是最多的而被刷掉了。所以我感觉上面的方案不行。这就考虑到“分而治之”中的“分”到底怎么分。。**在第二步中我们提到按照 IP 地址的 Hash(IP)%1024 的值，将海量 IP 日志分别存储到 1024 个小文件中**。。这样就会致使相似的 IP 或者同一 IP 被分到同一小文件中。。满足分而治之的要求。。故不存在分布均匀情况。。

还有一位网友给出了具体的方法：计数法（原理同上：分而治之）

假设一天之内某个 IP 访问百度的次数不超过 40 亿次,则访问次数可以用 unsigned 表示.用数组统计出每个 IP 地址出现的次数, 即可得到访问次数最大的 IP 地址。

IP 地址是 32 位的二进制数,所以共有 N=2^32=4G 个不同的 IP 地址, 创建一个 unsigned count[N];的数组,即可统计出每个 IP 的访问次数,而 sizeof(count) == 4G*4=16G, 远远超过了 32 位计算机所支持的内存大小,因此不能直接创建这个数组.下面采用划分法解决这个问题.

假设允许使用的内存是 512M, 512M/4=128M 即 512M 内存可以统计 128M 个不同的 IP 地址的访问次数.而 N/128M =4G/128M = 32 ,所以只要把 IP 地址划分成 32 个不同的区间,分别统计出每个区间中访问次数最大的 IP, 然后就可以计算出所有 IP 地址中访问次数最大的 IP 了.

因为 2^5=32, 所以可以把 IP 地址的最高 5 位作为区间编号, 剩下的 27 为作为区间内的值,建立 32 个临时文件,代表 32 个区间,把相同区间的 IP 地址保存到同一的临时文件中.

例如:

ip1=0x1f4e2342
ip1 的高 5 位是 id1 = ip1 >>27 = 0x11 = 3
ip1 的其余 27 位是 value1 = ip1 &0x07ffffff = 0x074e2342
所以把 value1 保存在 tmp3 文件中。
由 id1 和 value1 可以还原成 ip1, 即 ip1 =(id1<<27)|value1
按照上面的方法可以得到 32 个临时文件,每个临时文件中的 IP 地址的取值范围属于[0-128M),因此可以统计出每个

IP 地址的访问次数.从而找到访问次数最大的 IP 地址

程序源码:

```
#include <fstream>
#include <iostream>
#include <ctime>

using namespace std;
#define N 32            //临时文件数

#define ID(x)  (x>>27)                  //x 对应的文件编号
#define VALUE(x) (x&0x07ffffff)         //x 在文件中保存的值
#define MAKE_IP(x,y)  ((x<<27)|y)       //由文件编号和值得到 IP 地址.

#define MEM_SIZE  128*1024*1024        //需分配内存的大小为 MEM_SIZE*sizeof(unsigned)

char* data_path="D:/test/ip.dat";         //ip 数据

 //产生 n 个随机 IP 地址
void make_data(const int& n)
{
    ofstream out(data_path,ios::out|ios::binary);
    srand((unsigned)(time(NULL)));
    if (out)
    {
        for (int i=0; i<n; ++i)
        {
            unsigned val=unsigned(rand());
            val = (val<<24)|val;                //产生 unsigned 类型的随机数

            out.write((char *)&val,sizeof (unsigned));
        }
    }
}

//找到访问次数最大的 ip 地址
int main()
{
    //make_data(100);     //
    make_data(100000000);        //产生测试用的 IP 数据
    fstream arr[N];

    for (int i=0; i<N; ++i)                  //创建 N 个临时文件
    {
        char tmp_path[128];      //临时文件路径
        sprintf(tmp_path,"D:/test/tmp%d.dat",i);
        arr[i].open(tmp_path, ios::trunc|ios::in|ios::out|ios::binary);  //打开第 i 个文件

        if( !arr[i])
```

```
        {
            cout<<"open file"<<i<<"error"<<endl;
        }
    }

    ifstream infile(data_path,ios::in|ios::binary);   //读入测试用的 IP 数据
    unsigned data;

    while(infile.read((char*)(&data), sizeof(data)))
    {
        unsigned val=VALUE(data);
        int key=ID(data);
        arr[ID(data)].write((char*)(&val), sizeof(val));           //保存到临时文件件中
    }

    for(unsigned i=0; i<N; ++i)
    {
        arr[i].seekg(0);
    }
    unsigned max_ip = 0;    //出现次数最多的 ip 地址
    unsigned max_times = 0;     //最大只出现的次数

    //分配 512M 内存,用于统计每个数出现的次数
    unsigned *count = new unsigned[MEM_SIZE];

    for (unsigned i=0; i<N; ++i)
    {
        memset(count, 0, sizeof(unsigned)*MEM_SIZE);

        //统计每个临时文件件中不同数字出现的次数
        unsigned data;
        while(arr[i].read((char*)(&data), sizeof(unsigned)))
        {
            ++count[data];
        }

        //找出出现次数最多的 IP 地址
        for(unsigned j=0; j<MEM_SIZE; ++j)
        {
            if(max_times<count[j])
            {
                max_times = count[j];
                max_ip = MAKE_IP(i,j);         // 恢复成原 ip 地址.
            }
        }
    }
    delete[] count;
    unsigned char *result=(unsigned char *)(&max_ip);
    printf("出现次数最多的 IP 为:%d.%d.%d.%d,共出现%d 次",
        result[0], result[1], result[2], result[3], max_times);
```

```
}
```

运行结果：

2、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，这些查询串的重复读比较 高，虽然总数是 1 千万，但是如果去除重复和，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。

方案一：

分析：此问题的解决分为以下俩个步骤：
第一步：Query 统计
Query 统计有以下俩个方法，可供选择：
1）、直接排序法
首先我们最先想到的的算法就是排序了，首先对这个日志里面的所有 Query 都进行排序，然后再遍历排好序的 Query，统计每个 Query 出现的次数了。
但是题目中有明确要求，那就是内存不能超过 1G，一千万条记录，每条记录是 255Byte，很显然要占据 2.375G 内存，这个条件就不满足要求了。
让我们回忆一下数据结构课程上的内容，当数据量比较大而且内存无法装下的时候，我们可以采用外排序的方法来进行排序，这里我们可以采用归并排序，因为归并排序有一个比较好的时间复杂度 O(NlgN)。
排完序之后我们再对已经有序的 Query 文件进行遍历，统计每个 Query 出现的次数，再次写入文件中。
综合分析一下，排序的时间复杂度是 O(NlgN)，而遍历的时间复杂度是 O(N)，因此该算法的总体时间复杂度就是 O(N+NlgN)=O（NlgN）。

2）、Hash Table 法
在第 1 个方法中，我们采用了排序的办法来统计每个 Query 出现的次数，时间复杂度是 NlgN，那么能不能有更好的方法来存储，而时间复杂度更低呢？
题目中说明了，虽然有一千万个 Query，但是由于重复度比较高，因此事实上只有 300 万的 Query，每个 Query255Byte，因此我们可以考虑把他们都放进内存中去，而现在只是需要一个合适的数据结构，在这里，Hash Table 绝对是我们优先的选择，因为 Hash Table 的查询速度非常的快，几乎是 O(1)的时间复杂度。
那么，我们的算法就有了：维护一个 Key 为 Query 字串，Value 为该 Query 出现次数的 HashTable，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)的时间复杂度内完成了对该海量数据的处理。
本方法相比算法 1：在时间复杂度上提高了一个数量级，为 O（N），但不仅仅是时间复杂度上的优化，该方法只需要 IO 数据文件一次，而算法 1 的 IO 次数较多的，因此该算法 2 比算法 1 在工程上有更好的可操作性。

第二步：找出 Top 10
算法一：普通排序
我想对于排序算法大家都已经不陌生了，这里不在赘述，我们要注意的是排序算法的时间复杂度是 NlgN，在本题目中，三百万条记录，用 1G 内存是可以存下的。
算法二：部分排序
题目要求是求出 Top 10，因此我们没有必要对所有的 Query 都进行排序，我们只需要维护一个 10 个大小的数组，初始化

放入 10 个 Query，按照每个 Query 的统计次数由大到小排序，然后遍历这 300 万条记录，每读一条记录就和数组最小一个 Query 对比，如果小于这个 Query，那么继续遍历，否则，将数组中最后一条数据淘汰，加入当前的 Query（并寻找最小元素）。最后当所有的数据都遍历完毕之后，那么这个数组中的 10 个 Query 便是我们要找的 Top10 了。
不难分析出，这样，算法的最坏时间复杂度是 N*K， 其中 K 是指 top 多少。
算法三：堆
在算法二中，我们已经将时间复杂度由 NlogN 优化到 NK，不得不说这是一个比较大的改进了，可是有没有更好的办法呢？分析一下，在算法二中，每次比较完成之后，需要的操作复杂度都是 K，因为要把元素插入到一个线性表之中，而且采用的是顺序比较。这里我们注意一下，该数组是有序的，一次我们每次查找的时候可以采用二分的方法查找，这样操作的复杂度就降到了 logK，可是，随之而来的问题就是数据移动，因为移动数据次数增多了。不过，这个算法还是比算法二有了改进。
基于以上的分析，我们想想，有没有一种既能快速查找，又能快速移动元素的数据结构呢？回答是肯定的，那就是堆。
借助堆结构，我们可以在 log 量级的时间内查找和调整/移动。因此到这里，我们的算法可以改进为这样，维护一个 K(该题目中是 10)大小的小根堆，然后遍历 300 万的 Query，分别和根元素进行对比。
思想与上述算法二一致，只是算法在算法三，我们采用了最小堆这种数据结构代替数组，把查找目标元素的时间复杂度有 O（K）降到了 O（logK）。
那么这样，采用堆数据结构，算法三，最终的时间复杂度就降到了 N‘logK，和算法二相比，又有了比较大的改进。

总结：
至此，算法就完全结束了，经过上述第一步、先用 Hash 表统计每个 Query 出现的次数，O（N）；然后第二步、采用堆数据结构找出 Top 10，N*O（logK）。所以，我们最终的时间复杂度是：O（N） + N'*O（logK）。（N 为 1000 万，N’为 300 万）。

方案二：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。

分而治之 + hash 统计 + 堆/快速排序这个套路，我们已经开始有了屡试不爽的感觉。下面，再拿几道再多多验证下。请看此第 3 题：又是文件很大，又是内存受限，咋办?还能怎么办呢?无非还是：

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 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），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。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。

读者反馈@ylqndscylq：本文评论下，有读者 ylqndscylq 反应：每个小文件取前 100 会有问题。是否真如此，咱们先且看下一道题，第 4 题(稍后，我们将意识到，这第 3 题给出的算法有问题)。

有网友提出：呵呵

普通解法:分治，hash，归并，最大（小）堆，map reducer 等算法，你的小内存导致了只能用时间换空间的做法， 比如多次的遍历，big set 分裂成小 set，使用磁盘索引等。
2B 解法： lucene
文艺解法（ibm 研究院提供）：基于 priori algorithm.
http://rakesh.agrawal-family.com/papers/vldb94apriori.pdf

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。

方案 1：这题是考虑时间效率。用 trie 树统计每个词出现的次数，时间复杂度是 O(n*le)（le 表示单词的平准长度）。然后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。

5、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（最大数）。

此题与上面第 3 题类似，

1. 堆/归并排序：在每台电脑上求出 TOP10，可以采用包含 10 个元素的堆完成（TOP10 小，用最大堆，TOP10 大，用最小堆）。比如求 TOP10 大，我们首先取前 10 个元素调整成最小堆，如果发现，然后扫描后面的数据，并与堆顶元素比较，如果比堆顶元素大，那么用该元素替换堆顶，然后再调整为最小堆。最后堆中的元素就是 TOP10 大。

2. 求出每台电脑上的 TOP10 后，然后把这 100 台电脑上的 TOP10 组合起来，共 1000 个数据，再利用上面类似的方法求出 TOP10 就可以了。

读者反馈@herotangabc：这种在 n 个文件中找 top 几的算法明显是谬误的；我给你按照你这种方法举个简单例子就知道啦：比如求 2 个文件中的 top2，照你这种算法，如果第一个文件里有

a 49 次

b 50 次

c 2 次

d 1 次

第二个文件里有

a 9 次

b 1 次

c 11 次

d 10 次

那按照你的算法，第一个文件里出来 top2 是 b,a,第二个文件里出来 top2 是 c,d,然后 2 个 top2 归并，则算出所有的文件的 top2 是 b(50 次),a(49 次),但实际上是 a(58 次),b(51 次)。

@July 回馈：我想，这位读者可能没有明确题意。本题目中的 TOP10 是指最大的 10 个数，而不是指出现频率最多的 10 个数。但如果说，现在有另外一题，要你求频率最多的 10 个，相当于求访问次数最多的 10 个 IP 地址那道题，即是本文中上面的第 3 题。那么我的算法便是有问题的，也就是说，上述第 3 题的解法有误。特此说明。

6、100w 个数中找出最大的 100 个数。

方案 1：在前面的题中，我们已经提到了，用一个含 100 个元素的最小堆完成。复杂度为 O(100w*lg100)。
方案 2：采用快速排序的思想，每次分割之后只考虑比轴大的一部分，知道比轴大的一部分在比 100 多的时候，采用传统排序算法排序，取前 100 个。复杂度为 O(100w*100)。

算法如下：根据快速排序划分的思想
(1) 先对所有数据分成[a,b）b（b,d]两个区间，(b,d]区间内的数都是大于[a,b)区间内的数
(2) 对(b,d]重复(1)操作，直到最右边的区间个数小于 100 个。注意[a,b)区间不用划分
(3) 向左边的第一个区间取前 100-n.n 为已取出的元素个数。方法仍然是对其划分，取[c,d]区间。如果个数不够，继续(3)操作
(4) 有必要的话，对取出的 100 个数进行快速排序。over~

方案 3：采用局部淘汰法。选取前 100 个元素，并排序，记为序列 L。然后一次扫描剩余的元素 x，与排好序的 100 个元素中最小的元素比，如果比这个最小的要大，那么把这个最小的元素删除，并把 x 利用插入排序的思想，插入到序列 L 中。依次循环，知道扫描了所有的元素。复杂度为 O(100w*100)。

进一步：1 亿数据找出最大的 1w 个

1．分块法

解法：A. 采用分块法，将 1 亿数据分成 100w 一块，共 100 块。

B. 对每块进行快速排序，分成两堆，如果大堆大于 1w 个，则对大堆再次进行快速排序，直到小于等于 1w 停止

（假设此时大堆有 N 个），此时对小堆进行排序，取最大的 10000-N 个，这样就找到了这 100w 中最大的 1w 个。

C. 100 块，每块选出最大的 1w，再对这 100w 使用同样的方法，找出最大的 1w 个

2. 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

解法：用一个例子来说明吧，这样直观一点。

假设对 7, 6, 3, 5 这四个数进行排序，首先初始化一个 byte，8 位，可表示为 0 0 0 0 0 0 0 0

对于 7，将第七位置 1，对剩下几个数执行同样操作，则最后该 byte 变为 0 0 1 0 1 1 1 0

最后一步，遍历，将置 1 位的序号逐个输出，即 3，5， 6，7

3. 红黑树

解法：用一个红黑树维护这 1w 个数，然后遍历其他数字，来替换红黑树中最小的数（这是在网上看到的算法，

我感觉用赢 者树也是可以的）

如果数据中有重复，则对于 Bit-Map，找出前 1w 个数，对这 1w 个数建立 Hash Table，然后再次遍历这一亿个数，同时对 Hash Table 中的数字 计数，最后根据计数找出前 1w 个（包含重复）

7、有 10 个文件，每个文件 1G，每个文件的每一行存放的都是用户的 query，每个文件的 query 都可能重复。要求你按照 query 的频度排序。

直接上：

1. hash 映射：顺序读取 10 个文件，按照 hash(query)%10 的结果将 query 写入到另外 10 个文件（记为）中。这样新生成的文件每个的大小大约也 1G（假设 hash 函数是随机的）。
2. hash 统计：找一台内存在 2G 左右的机器，依次对用 hash_map(query, query_count)来统计每个 query 出现的次数。注：hash_map(query,query_count)是用来统计每个 query 的出现次数，不是存储他们的值，出现一次，则 count+1。
3. 堆/快速/归并排序：利用快速/堆/归并排序按照出现次数进行排序。将排序好的 query 和对应的 query_cout 输出到文件中。这样得到了 10 个排好序的文件（记为）。对这 10 个文件进行归并排序（内排序与外排序相结合）。

除此之外，此题还有以下两个方法：

方案 2：一般 query 的总量是有限的，只是重复的次数比较多而已，可能对于所有的 query，一次性就可以加入到内存了。这样，我们就可以采用 trie 树/hash_map 等直接来统计每个 query 出现的次数，然后按出现次数做快速/堆/归并排序就可以了。

方案 3：与方案 1 类似，但在做完 hash，分成多个文件后，可以交给多个文件来处理，采用分布式的架构来处理（比如 MapReduce），最后再进行合并。

8、 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？

方案一：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。

1. 分而治之/hash 映射：遍历文件 a，对每个 url 求取 $\text{hash}(url)\%1000$ ，然后根据所取得的值将 url 分别存储到 1000 个小文件（记为 $a_0, a_2, \cdots, a_{999}$ ）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 小文件中（记为 $b_0, b_1, \cdots, b_{999}$ ）。这样处理后，所有可能相同的 url 都在对应的小文件（a_0 vs b_0, a_1 vs b_1, …, a_0 vs b_{999}）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。
2. hash 统计：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。

OK，此第一种方法：分而治之/hash 映射 + hash 统计 + 堆/快速/归并排序，再看最后三道题，如下：

方案二：如果允许有一定的错误率，可以使用 Bloom filter，4G 内存大概可以表示 340 亿 bit。将其中一个文件中的 url 使用 Bloom filter 映射为这 340 亿 bit，然后挨个读取另外一个文件的 url，检查是否与 Bloom filter，如果是，那么该 url 应该是共同的 url（注意会有一定的错误率）。

9、怎么在海量数据中找出重复次数最多的一个？

方案 1：先做 hash，然后求模映射为小文件，求出每个小文件中重复次数最多的一个，并记录重复次数。然后找出上一步求出的数据中重复次数最多的一个就是所求（具体参考前面的题）。

10、上千万或上亿数据（有重复），统计其中出现次数最多的钱 N 个数据。

方案 1：上千万或上亿的数据，现在的机器的内存应该能存下。所以考虑采用 hash_map/搜索二叉树/红黑树等来进行统计次数。然后就是取出前 N 个出现次数最多的数据了，可以用第 2 题提到的堆机制完成。

11、 1000 万字符串，其中有些是重复的，需要把重复的全部去掉，保留没有重复的字符串。请怎么设计和实现？

- 方案 1：这题用 trie 树比较合适，hash_map 也行。

首先映射为内存可以处理的 n 个小文件，这时相同的字符串肯定在同一个文件中，在每个小文件中使用 hash_set 取出重复的字符串，之后写 到一个文件中，依次处理 n 个文件，即可得到结果。。

- 方案 2：from xjbzju:，1000w 的数据规模插入操作完全不现实，以前试过在 stl 下 100w 元素插入 set 中已经慢得不能忍受，觉得基于 hash 的实现不会比红黑树好太多，使用 vector+sort+unique 都要可行许多，建议还是先 hash 成小文件分开处理再综合。

上述方案 2 中读者 xbzju 的方法让我想到了一些问题，即是 set/map，与 hash_set/hash_map 的性能比较?共计 3 个问题，如下：

- 1、hash_set 在千万级数据下，insert 操作优于 set? 这位 blog：http://t.cn/zOibP7t 给的实践数据可靠不?
- 2、那 map 和 hash_map 的性能比较呢？谁做过相关实验?

```
set VS hash_set VS hash_table(强化版) 性能测试
数据容量 10000000个 查询次数 10000000次
容器中数据范围 [0, 40000000) 查询数据范围[0, 40000000)
--by MoreWindows( http://blog.csdn.net/MoreWindows ) --

-----插入数据-----------
set中有数据8061105个
set 的 insert操作 用时 18782毫秒
hash_set中有数据8061105个
hash_set 的 insert操作 用时 7722毫秒
hash_table中有数据8061105个
Hash_table 的 insert操作 用时 4930毫秒
```

- 3、那查询操作呢，如下段文字所述?

可以发现在hash_table中最长的链表也只有5个元素，**长度为1和长度为2的链表中的数据占全部数据的89%以上。因此绝大数查询将仅仅访问哈希表1次到2次。这样的查询效率当然会比set（内部使用红黑树——类似于二叉平衡树）高的多。**有了这个图示，无疑已经可以证明hash_set会比set快速高效了。但hash_set还可以动态的增加表的大小，因此我们再实现一个表大小可增加的hash_table。

或者小数据量时用 map，构造快，大数据量时用 hash_map?

rbtree PK hashtable

据朋友№邦卡猫№的做的红黑树和 hash table 的性能测试中发现：当数据量基本上 int 型 key 时，hash table 是 rbtree 的 3-4 倍，但 hash table 一般会浪费大概一半内存。

因为 hash table 所做的运算就是个%，而 rbtree 要比较很多，比如 rbtree 要看 value 的数据 ，每个节点要多出 3 个指针（或者偏移量） 如果需要其他功能，比如，统计某个范围内的 key 的数量，就需要加一个计数成员。

且 1s rbtree 能进行大概 50w+次插入，hash table 大概是差不多 200w 次。不过很多的时候，其速度可以忍了，例如倒排索引差不多也是这个速度，而且单线程，且倒排表的拉链长度不会太大。正因为基于树的实现其实不比 hashtable 慢到哪里去，所以数据库的索引一般都是用的 B/B+树，而且 B+树还对磁盘友好(B 树能有效降低它的高度，所以减少磁盘交互次数)。比如现在非常流行的 NoSQL 数据库，像 MongoDB 也是采用的 B 树索引。关于 B 树系列，请参考本 blog 内此篇文章：从 B 树、B+树、B*树谈到 R 树。

OK，更多请待后续实验论证。接下来，咱们来看第二种方法，双层捅划分。

密匙二、双层桶划分

双层桶划分----其实本质上还是分而治之的思想，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

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：

11、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。

```
#include<stdio.h>
#include<memory.h>
//用 char 数组存储 2-Bitmap,不用考虑大小端内存的问题
unsigned char flags[1000]; //数组大小自定义

unsigned get_val(int idx)
{
    int i = idx/4;
    int j = idx%4;
    unsigned ret = (flags[i]&(0x3<<(2*j)))>>(2*j);
    return ret;
}

unsigned set_val(int idx, unsigned int val)
{
    int i = idx/4;
    int j = idx%4;
    unsigned tmp = (flags[i]&~((0x3<<(2*j))&0xff)) | (((val%4)<<(2*j))&0xff);
    flags[i] = tmp;
    return 0;
}
unsigned add_one(int idx)
{
    if (get_val(idx)>=2) {
        return 1;
    }
    else  {
        set_val(idx, get_val(idx)+1);
        return 0;
    }
}

//只测试非负数的情况;
//假如考虑负数的话,需增加一个 2-Bitmap 数组.
int a[]={1, 3, 5, 7, 9, 1, 3, 5, 7, 1, 3, 5,1, 3, 1,10,2,4,6,8,0};

int main()
{
    int i;
    memset(flags, 0, sizeof(flags));

    printf("原数组为:");
    for(i=0;i < sizeof(a)/sizeof(int); ++i)  {
        printf("%d  ", a[i]);
        add_one(a[i]);
    }
```

```
    printf("\r\n");

    printf("只出现过一次的数:");
    for(i=0;i < 100; ++i)  {
        if(get_val(i) == 1)
            printf("%d  ", i);
    }
    printf("\r\n");

    return 0;
}
```

除了用 2-Bitmap 来计数标记以外,也可以用两个 1-Bitmap 来实现(如果考虑正负数的情况,就四个 1-Bitmap)

12、5 亿个 int 找它们的中位数。

1. 思路一：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。
2. 思路二@绿色夹克衫：同样需要做两遍统计，如果数据存在硬盘上，就需要读取 2 次。
   方法同基数排序有些像，开一个大小为 65536 的 Int 数组，第一遍读取，统计 Int32 的高 16 位的情况，也就是 0-65535，都算作 0,65536 - 131071 都算作 1。就相当于用该数除以 65536。Int32 除以 65536 的结果不会超过 65536 种情况，因此开一个长度为 65536 的数组计数就可以。每读取一个数，数组中对应的计数+1，考虑有负数的情况，需要将结果加 32768 后，记录在相应的数组内。
   第一遍统计之后，遍历数组，逐个累加统计，看中位数处于哪个区间，比如处于区间 k，那么 0- k-1 的区间里数字的数量 sum 应该<n/2（2.5 亿）。而 k+1 - 65535 的计数和也<n/2，第二遍统计同上面的方法类似，但这次只统计处于区间 k 的情况，也就是说(x / 65536) + 32768 = k。统计只统计低 16 位的情况。并且利用刚才统计的 sum，比如 sum = 2.49 亿，那么现在就是要在低 16 位里面找 100 万个数(2.5 亿-2.49 亿)。这次计数之后，再统计一下，看中位数所处的区间，最后将高位和低位组合一下就是结果了。

密匙三：Bloom filter/Bitmap

Bloom filter

关于什么是 **Bloom filter**，请参看 blog 内此文：

- 海量数据处理之 Bloom Filter 详解

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。

13、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。

同时，上文的第 5 题：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？如果允许有一定的错误率，可以使用 Bloom filter，4G 内存大概可以表示 340 亿 bit。将其中一个文件中的 url 使用 Bloom filter 映射为这 340 亿 bit，然后挨个读取另外一个文件的 url，检查是否与 Bloom filter，如果是，那么该 url 应该是共同的 url（注意会有一定的错误率）。

Bitmap

- 关于什么是 Bitmap，请看 blog 内此文第二部分：http://blog.csdn.net/v_july_v/article/details/6685962。

下面关于 Bitmap 的应用，直接上题，如下第 9、10 道：

14/11 题、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。

方案 1：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。

方案 2：也可采用与第 1 题类似的方法，进行划分小文件的方法。然后在小文件中找出不重复的整数，并排序。然后再进行归并，注意去除重复的元素。

15、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？

第一反应时快速排序+二分查找。以下是其它更好的方法：

方案 1：frome oo，用位图/Bitmap 的方法，申请 512M 的内存，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：这个问题在《编程珠玑》里有很好的描述，大家可以参考下面的思路，探讨一下：
又因为 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)，方案 2 完。

16、给 40 亿个 unsigned int 的整数，如何判断这 40 亿个数中哪些数重复？

同理，可以申请 512M 的内存空间，然后读取 40 亿个整数，并且将相应的 bit 位置 1。如果是第一次读取某个数据，则在将该 bit 位置 1 之前，此 bit 位必定是 0；如果是第二次读取该数据，则可根据相应的 bit 位是否为 1 判断该数据是否重复。

附：这里，再简单介绍下，位图方法：
使用位图法判断整形数组是否存在重复
判断集合中存在重复是常见编程任务之一，当集合中数据量比较大时我们通常希望少进行几次扫描，这时双重循环法就不可取了。

位图法比较适合于这种情况，它的做法是按照集合中最大元素 max 创建一个长度为 max+1 的新数组，然后再次扫描原数组，遇到几就给新数组的第几位置上 1，如遇到 5 就给新数组的第六个元素置 1，这样下次再遇到 5 想置位时发现新数组的第六个元素已经是 1 了，这说明这次的数据肯定和以前的数据存在着重复。这 种给新数组初始化时置零其后置一的做法类似于位图的处理方法故称位图法。它的运算次数最坏的情况为 2N。如果已知数组的最大值即能事先给新数组定长的话效 率还能提高一倍。

密匙四、Trie 树/数据库/倒排索引

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. 上面的**第 2 题**：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。
2. 上面的**第 5 题**：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. 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？
4. 上面的**第 8** 题：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。其解决方法是：用 trie 树统计每个词出现的次数，时间复杂度是 O(n*le)（le 表示单词的平准长度），然后是找出出现最频繁的前 10 个词。

更多有关 Trie 树的介绍，请参见此文：从 Trie 树（字典树）谈到后缀树。

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：利用数据的设计实现方法，对海量数据的增删改查进行处理。

- 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://www.codinglabs.org/html/theory-of-mysql-index.html；
- 关于 B 树、B+ 树、B* 树及 R 树，本 blog 内有篇绝佳文章：http://blog.csdn.net/v_JULY_v/article/details/6530142。

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。

关于倒排索引的应用，更多请参见：

- 第二十三、四章：杨氏矩阵查找，倒排索引关键词 Hash 不重复编码实践，
- 第二十六章：基于给定的文档生成倒排索引的编码与实践。

密匙五、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 明显不够，所以可以用来排序。内存可以当输入缓冲区使用。

关于多路归并算法及外排序的具体应用场景，请参见 blog 内此文：

- 第十章、如何给 10^7 个数据量的磁盘文件排序

密匙六、分布式处理之 Mapreduce

MapReduce 是一种计算模型，简单的说就是将大批量的工作（数据）分解（MAP）执行，然后再将结果合并成最终结果（REDUCE）。这样做的好处是可以在任务被分解后，可以通过大量机器进行并行计算，减少整个操作的时间。但如果你要我再通俗点介绍，那么，说白了，Mapreduce 的原理就是一个归并排序。

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:
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)？

更多具体阐述请参见 blog 内：

- 从 Hadhoop 框架与 MapReduce 模式中谈海量数据处理，
- 及 MapReduce 技术的初步了解与学习。

其它模式/方法论，结合操作系统知识

至此，六种处理海量数据问题的模式/方法已经阐述完毕。据观察，这方面的面试题无外乎以上一种或其变形，然题目为何取为是：秒杀 99%的海量数据处理面试题，而不是 100%呢。OK，给读者看最后一道题，如下：

非常大的文件，装不进内存。每行一个 int 类型数据，现在要你随机取 100 个数。

我们发现上述这道题，无论是以上任何一种模式/方法都不好做，那有什么好的别的方法呢？我们可以看看：操作系统内存分页系统设计(说白了，就是映射+建索引)。

Windows 2000 使用基于分页机制的虚拟内存。每个进程有 4GB 的虚拟地址空间。基于分页机制，这 4GB 地址空间的一些部分被映射了物理内存，一些部分映射硬盘上的交换文 件，一些部分什么也没有映射。程序中使用的都是 4GB 地址空间中的虚拟地址。而访问物理内存，需要使用物理地址。 关于什么是物理地址和虚拟地址，请看：

- 物理地址 (physical address)：放在寻址总线上的地址。放在寻址总线上，如果是读，电路根据这个地址每位的值就将相应地址的物理内存中的数据放到数据总线中传输。如果是写，电路根据这个 地址每位的值就将相应地址的物理内存中放入数据总线上的内容。物理内存是以字节(8 位)为单位编址的。
- 虚拟地址 (virtual address)：4G 虚拟地址空间中的地址，程序中使用的都是虚拟地址。 使用了分页机制之后，4G 的地址空间被分成了固定大小的页，每一页或者被映射到物理内存，或者被映射到硬盘上的交换文件中，或者没有映射任何东西。对于一 般程序来说，4G 的地址空间，只有一小部分映射了物理内存，大片大片的部分是没有映射任何东西。物理内存也被分页，来映射地址空间。对于 32bit 的 Win2k，页的大小是 4K 字节。CPU 用来把虚拟地址转换成物理地址的信息存放在叫做页目录和页表的结构里。

物理内存分页，一个物理页的大小为 4K 字节，第 0 个物理页从物理地址 0x00000000 处开始。由于页的大小为 4KB，就是 0x1000 字节，所以第 1 页从物理地址 0x00001000 处开始。第 2 页从物理地址 0x00002000 处开始。可以看到由于页的大小是 4KB，所以只需要 32bit 的地址中高 20bit 来寻址物理页。

返回上面我们的题目：非常大的文件，装不进内存。每行一个 int 类型数据，现在要你随机取 100 个数。针对此题，我们可以借鉴上述操作系统中内存分页的设计方法，做出如下解决方案：

操作系统中的方法，先生成 4G 的地址表，在把这个表划分为小的 4M 的小文件做个索引，二级索引。30 位前十位表示第几个 4M 文件，后 20 位表示在这个 4M 文件的第几个，等等，基于 key value 来设计存储，用 key 来建索引。

但如果现在只有 10000 个数，然后怎么去随机从这一万个数里面随机取 100 个数？请读者思考。更多海里数据处理面试题，请参见此文第一部分：http://blog.csdn.net/v_july_v/article/details/6685962。

当一个应用的数据量大的时候，我们用单表和单库来存储会严重影响操作速度，如 mysql 的 myisam 存储，我们经过测试，200w 以下的时候，mysql 的访问速度都很快，但是如果超过 200w 以上的数据，他的访问速度会急剧下降，影响到我们 webapp 的访问速度，而且数据量太大的话，如果用单表存储，就会使得系统相当的不稳定，mysql 服务很容易挂掉。所以当数据量超过 200w 的时候，建议系统工程师还是考虑分表.

以下是几种常见的分表算法：

1.按自然时间来分表/分库;

如一个应用的数据在一年后数据量会达到 200w 左右，那么我们就可以考虑用一年的数据来做为一个表或者库来存储，例如，表名为 app，那么 2010 年的数据就是 app_2010，app_2011;如果数据量在一个月就达到了 200w 左右，那么我们就可以用月份来分，app_2010_01，app_2010_02.

2.按数字类型 hash 分表/分库;

如果我们要存储用户的信息，我们应用的注册量很大，我们用单表是不能满足存储需求的，那么我们就可以用用户的编号来进行 hash，常见的是用取余操作，如果我们要分 30 张表来存储用户的信息，那么用户编号为 1 的用户 1%30=1，那么我们就存在 user_01 表里，如用户的编号为 500，那么 500%30=20，那么我们就将此用户的信息存储在 user_20 的表里.

3.按 md5 值来分表/分库;

我们假设要存储用户上传的文件，如果上传量大的话，也会带来系统的瓶颈问题，我们做过试验，在一个文件夹下如果超过 200 个文件的话，文件的浏览效率会降低，当然，这个不属于我们本文讨论的范围，这块也要做散列操作.我们可以用文件的用户名来 md5 或者用文件的 md5 校验值来做，我们就可以用 md5 的前 5 位来做 hash，这样最多我们就可以得到 5^5=3125 个表，每次在存储文件的时候，就可以用文件名的 md5 值的前 5 位来确定这个文件该存那张表.

实例:某微博的 url 加密算法和存储策略的猜想

现在好多微博都用这样的 url 来访问，如果他们的域名为 www.example.com，那么如果你发微博的时候，你会发现你所发的 url 都变成了 http://t.cn/Mx4ja1，这样的形式，他们是怎么进行这样的转换呢?我猜想就是用到了我们上面讲的 md5 的存储和查找规则，用你发的 url 来进行 md5，得到 md5 值之后，如我们例子来说，就会用前 6 位来进行分表.

分表所带来的问题

分表也会带来一系列的问题，如分页的实现，统计的实现，如果我们要做一个所有数据的分页，那么我们得每张表都得遍历一遍，这样访问效率会很低下.之前我尝试过用 mysql 的代理来实现，最终用 tcsql 来实现了.

分表算法的选择

首先，分表适合于没有大的列表的应用来使用，要不然，会为这部分做好多额外的工作，如果你的应用数据量不是特别大的话，最好别用分表。呵呵，以前在做项目的时候，一项目经理要我们设计了一个千万级别的分表算法，而应用的 pv 不会超过 100，总有点大炮打蚊子的感觉，而且因为分表，把整个项目的工期拖延了不少，得不偿失。

题目：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。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）。

关于中位数：数据排序后，位置在最中间的数值。即将数据分成两部分，一部分大于该数值，一部分小于该数值。中位数的位置：当样本数为奇数时，中位数=(N+1)/2；当样本数为偶数时，中位数为 N/2 与 1+N/2 的均值（那么 10G 个数的中位数，就第 5G 大的数与第 5G+1 大的数的均值了）。

分析：明显是一道工程性很强的题目，和一般的查找中位数的题目有几点不同。

1. 原数据不能读进内存，不然可以用快速选择，如果数的范围合适的话还可以考虑桶排序或者计数排序，但这里假设是 32 位整数，仍有 4G 种取值，需要一个 16G 大小的数组来计数。

2. 若看成从 N 个数中找出第 K 大的数，如果 K 个数可以读进内存，可以利用最小或最大堆，但这里 K=N/2,有 5G 个数，仍然不能读进内存。

3. 接上，对于 N 个数和 K 个数都不能一次读进内存的情况，《编程之美》里给出一个方案：设 k<K,且 k 个数可以完全读进内存，那么先构建 k 个数的堆，先找出第 0 到 k 大的数，再扫描一遍数组找出第 k+1 到 2k 的数，再扫描直到找出第 K 个数。虽然每次时间大约是 nlog(k)，但需要扫描 ceil(K/k) 次，这里要扫描 5 次。

解法：首先假设是 32 位无符号整数。

1. 读一遍 10G 个整数，把整数映射到 256M 个区段中，用一个 64 位无符号整数给每个相应区段记数。

说明：整数范围是 0 - 2^32 - 1，一共有 4G 种取值，映射到 256M 个区段，则每个区段有 16（4G/256M = 16）种值，每 16 个值算一段， 0～15 是第 1 段，16～31 是第 2 段，……2^32-16 ～2^32-1 是第 256M 段。一个 64 位无符号整数最大值是 0～8G-1，这里先不考虑溢出的情况。总共占用内存 256M×8B=2GB。

2. 从前到后对每一段的计数累加，当累加的和超过 5G 时停止，找出这个区段（即累加停止时达到的区段，也是中位数所在的区段）的数值范围，设为[a，a+15]，同时记录累加到前一个区段的总数，设为 m。然后，释放除这个区段占用的内存。

3. 再读一遍 10G 个整数，把在[a，a+15]内的每个值计数，即有 16 个计数。

4. 对新的计数依次累加，每次的和设为 n，当 m+n 的值超过 5G 时停止，此时的这个计数所对应的数就是中位数。

总结：

1.以上方法只要读两遍整数，对每个整数也只是常数时间的操作，总体来说是线性时间。

2. 考虑其他情况。

若是有符号的整数，只需改变映射即可。若是 64 为整数，则增加每个区段的范围，那么在第二次读数时，要考虑更多的计数。若过某个计数溢出，那么可认定所在的区段或代表整数为所求，这里只需做好相应的处理。噢，忘了还要找第 5G+1 大的数了，相信有了以上的成果，找到这个数也不难了吧。

3. 时空权衡。

花费 256 个区段也许只是恰好配合 2GB 的内存（其实也不是，呵呵）。可以增大区段范围，减少区段数目，节省一些内存，虽然增加第二部分的对单个数值的计数，但第一部分对每个区段的计数加快了（总体改变？？待测）。

4. 映射时尽量用位操作，由于每个区段的起点都是 2 的整数幂，映射起来也很方便。

2013 网易校园招聘笔试题

1、假设进栈次序是 e1, e2, e3, e4，那可能的出栈次序是()

A、e2, e4, e3, e1

B、e2, e3, e4, e1

C、e3, e2, e4, e1

D、e1, e2, e4, e3

给定入栈顺序，求出可能的出栈顺序。（点评：老得掉渣得题目了，只要小心点都没有问题）

2、表达式 X=A+B*(C-D)/E 的后缀表示形式可以是()

A、XAB+CDE/-*=

B、XA+BC-DE/*=

C、XABCD-*E/+=
D、XABCDE+*/=
分析：XABCD-*E/+=
3.四叉树中包含地空指针数量有多少？假设每个节点含有四个指向其孩子的指针，那么给定 n 个节点，其 4n 个指针有多少指向空？（比较简单的题目，n 个节点使用了的指针有 n－1,所以最后的答案位 4n－（n－1）＝3n＋1）
分析：或者举例说明也行。。
4.那个排序算法是非稳定的？选择，冒泡、希尔，堆排序，快速等 （也是比较基础的题目）
A、冒泡排序　　B、归并排序　　C、快速排序　　D、堆排序　　E、希尔排序
分析：凡是 O(n^2)的全部是稳定排序，O(nlogn)的全部是非稳定排序。。

排序效率比较

排序法	最差时间分析	平均时间复杂度	稳定度	空间复杂度
冒泡排序	$O(n^2)$	$O(n^2)$	稳定	O(1)
快速排序	$O(n^2)$	$O(n*\log_2 n)$	不稳定	$O(\log_2 n)\sim O(n)$
选择排序	$O(n^2)$	$O(n^2)$	稳定	O(1)
二叉树排序	$O(n^2)$	$O(n*\log_2 n)$	不一顶	O(n)
插入排序	$O(n^2)$	$O(n^2)$	稳定	O(1)
堆排序	$O(n*\log_2 n)$	$O(n*\log_2 n)$	不稳定	O(1)
希尔排序	0	0	不稳定	O(1)

5.根据函数，赋予参数值，写输出。。请问 func（0x7f530829）的返回值是（）

```
int func(unsigned int i)
{
    unsigned int temp = i;
    temp = (temp & 0x55555555) + ((temp & 0xaaaaaaaa)>>1);
    temp = (temp & 0x33333333) + ((temp & 0xcccccccc)>>2);
    temp = (temp & 0x0f0f0f0f) + ((temp & 0xf0f0f0f0)>>4);
    temp = (temp & 0xff00ff) + ((temp & 0xff00ff00)>>8);
    temp = (temp & 0xffff) + ((temp & 0xffff0000)>>16);
    return temp;
}
```

A、15　　B、16　　C、17　　D、18
分析：函数实现的是求二进制表示的时候，1 的个数，一共 15 个
最开始把每一个位看做一个节点，相邻节点值相加，结果用两个位表示。。。
然后每两个位看做一个节点，相邻节点值相加，结果用四个位表示。。。
以此类推，直到只剩下一个节点。。。
6.进程与线程的区别：系统调度是对进程还是线程，线程与进程共享的内存空间、公共地址空间等；
A.操作系统只调度进程，不调度线程
B.线程共享内存地址空间，进程不共享
C.线程间可共享内存数据，但进程不可以

D.进程可以通过 IPC 通信,但线程不可以

7.内存管理：段页式管理，地址映射表是？（操作系统方面的知识也不能掉以轻心呀）

A. 每个作业或进程一张段表，一张页表

B. 每个作业或进程的每个段一张段表，一张页表

C. 每个作业或进程一张段表，每个段一张页表

D. 每个作业一张页表，每个段一张段表

8、关于 TCP 协议，下面哪种说法是错误的（）

A、TCP 关闭连接过程中，两端的 socket 都会经过 TIME_WAIT 状态

B、对一个 Established 状态的 TCP 连接，调用 shutdown 函数可以让主动调用的一方进入半关闭状态

C、TCP 协议默认保证了当 TCP 的一端发生意外崩溃（当机、网线断开或路由器故障），另一端能自动检测到连接失效

D、在成功建立连接的 TCP 上，只有在 Established 状态才能收发数据，其他状态都不可以。

分析：tcp/ip 协议的实际使用过程中的问题：例如单方面断开后，另一端出于哪种状态，还有

9、关于主键 Primary Key 和索引 index 的说法哪些是错误的？（）

A、唯一索引的列允许为 NULL 值

B、一个关系表中的外键必定是另一表中的主键

C、一个表中只能有一个唯一性索引

D、索引主要影响查询过程，对数据的插入影响不大

分析：数据库方面的知识：主键和索引的基本定义及其性质，例如主键在表中是否唯一，索引的速度以及对表的改变的影响；无论是唯一索引还是非唯一索引，索引列都允许取 NULL 值

10、数据库的事务隔离级别一般分为 4 个级别，其中可能发生“不可重复读”的事物级别有（）

A、SERIALIZABLE

B、READ COMMITTED

C、READ UNCOMMITTED

D、REPEATABLE READ

分析数据库：数据库的不可重复访问异常，四种事务隔离级别中哪些可以避免该类异常？

各隔离级别对各种异常的控制能力

LU 丢失更新 DR 脏读 NRR 非重复读 SLU 二类丢失更新 PR 幻像读未提交读 RUYYYYY 提交读 RCNNYYY 可重复读 RRNNNNY 串行读 SNNNNY

11、如果 F(n)为该数列的第 n 项，那么这句话可以写成如下形式：

F(1)=1，F(2)=1，F(n)=F(n-1)+F(n-2) (n>=3)

请实现该函数 F(n)的求解，并给出算法复杂度，要求算法复杂度小于 O(n^2)。

思路：使用滚动数组可以保存以前保存的结果，加快速度，减少空间复杂度。

```
int Fib(int index)
{
	if(index<1)
	{
		return-1;
	}
	int a1=1,a2=1,a3=1;
	for(int i=0;i<index-2;i++)
	{
		a3=a1+a2;
		a1=a2;
		a2=a3;
	 }
	 return a3;
}
```

详见:菲波那切数列七种解法:http://www.cnblogs.com/hlxs/archive/2011/07/15/2107389.html
第二 部分（必做）：程序设计
1、下面的程序的输出是什么？

```
#include<stdio.h>int main(void)
{
    int n;
    char y[10] = "ntse";
    char *x = y;
    n = strlen(x);
    *x = x[n];
    x++;
    printf("x=%s\n",x);
    printf("y=%s\n",y);
    return 0;
}
```

输出：
x=tse
y=
因为 n=4，则*x = x[n]； 的功能是将 x 指向的第一个字符 n 修改为\0，这样 y 字符串就结束了，所以第二输出为空，x++操作后，x 指向第二个字符 t，所以第一个输出为：tse。
2、请给出下面程序的输出结果，并说明原因。

```
#include<iostream>#include<vector>using namespace std;
template<class t>class array
{public:
    array(int size);
    size_t getVectorSize()
    {
        return _data.size();
    }
    size_t getSize()
    {
        return _size;
    }public:
    vector<t> _data;
    size_t _size;
};

template<class t>array<t>::array(int size) : _size(size) , _data(_size)
{    }int main(void)
{
    array<int> *arr = new array<int>(3);
    cout<<arr->getVectorSize()<<endl;
    cout<<arr->getSize()<<endl;
    return 0;
}
```

12.写一个程序来确定系统是大端模式还是小端模式；
13.编程实现采用位操作来实现整数的加法操作。
14. 图的矩阵表示法，图的深度优先遍历，算法思路及其实现。
15.CAS（compare and swap）操作实现：（具体原理可以参考）
16.fork 函数的用法。具体题目为：

```
#include <stdio.h>#include <sys/types.h>#include <unistd.h>
int main(void)
{
    int i;
    for(i=0; i<2; i++){
        fork();
        printf("-");
        fflush(stdout);
    }

    return 0;
}
```

6 个-
详见：http://coolshell.cn/articles/7965.html
17.spin lock 原理：
先来一些代码吧！

```
void initlock（volatile int* lock_status）
{
    *lock_status = 0;
}void lock（volatile int* lock_status）
{
    while(test_and_set(lock_status = =1);
}void unlock(volatile int* lock_status)
{
    *lock_status = 0;
}
```

问题：volatile 的作用？lock 函数优化（针对在多 cpu 上提高 cpu cache）？上面的缺陷（内存模式上的）？
volatile 的作用： 作为指令关键字，确保本条指令不会因编译器的优化而省略，且要求每次直接读值。如果没有 volatile，基本上会导致这样的结果：要么无法编写多线程程序，要么编译器失去大量优化的机会。
18.给定一个巨大的文件，如何从中选出 k 行，随处输出 k 行到文件中。要求每一行出现的概率都相等。设计算法、说明思路，算法复杂度。
19.win32 中 WM_Quit 的作用是什么？
20.比较 mutex 和临街区之间的区别，并说明其使用场景。
21.多线程编程，如何安全退出线程。
还有网易数据挖掘方面的题目，这次数据挖掘的题目比较新奇，都是简答题。如下：
1，简述你对数据与处理的认识；
2，简述你对中文分词的理解，说明主要难点和常用算法；
3，常见的分类算法有哪些；
4，简述 K-MEANS 算法；
5，设计一个智能的商品推荐系统；
6，简述你对观点挖掘的认识

网易游戏笔试的人太少，因此可提供的笔试题目都不全，只是听说特别的难。还有好多是数学方面的智力题。例如：
1、英雄升级，从 0 级升到 1 级，概率 100%。
从 1 级升到 2 级，有 1/3 的可能成功；1/3 的可能停留原级；1/3 的可能下降到 0 级；
从 2 级升到 3 级，有 1/9 的可能成功；4/9 的可能停留原级；4/9 的可能下降到 1 级。
每次升级要花费一个宝石，不管成功还是停留还是降级。
求英雄从 0 级升到 3 级平均花费的宝石数目。

2012 网易校园招聘笔试题

第一部分（必做）：计算机科学基础
1、长为 N 的字符串中匹配长度为 M 的子串的算法复杂度是（）
A. O(N) B. O(M+N) C. O(N+logM) D. O(M+logN)
答：B
分析：我查了查，O(M + N)。KMP 能做到。
这里：http://blog.csdn.net/meixr/article/details/6456896
2、以下排序算法中，哪些是稳定的排序算法（多选）（）
A.冒泡 B.插入 C.合并 D.希尔 E.快速排序
答：ABC
3、以下是一颗平衡二叉树，请画出插入键值 3 以后的这颗平衡二叉树。
分析：考察平衡二叉树的基本操作，插入 3 变成不平衡，需要节点 5 右旋一次，节点 2 左旋一次。。

4、给定两个整数集合 A 和 B，每个集合都包含 20 亿个不同整数，请给出快速计算 A∩B 的算法，算法可使用外存，但是要求占用内存不能超过 4GB。
答： 将集合 A 是的整数，根据 n%10 不同，分别装入 10 个文件中，依次命名为 a0,a1……,a9。同理，将集合 B 分别装入 10 个文件中，依次命名为 b0,b1,……,b9。那么 A 和 B 编号不同的文件中，一定不会有相同的整数。只需分另求出 a0 与 b0 中共有的元素、a1 与 b1 中共有的元素……
利用 bitmap，将 bitmap 清 0，读入文件 ai，依次处理每个数，即将 bitmap 的第（n/10）位置 1。然后读入文件 bi，依次处理每个数，即：若 bitmap 第（n/10）位为 1，则这个数属于 A∩B
5、请给出从 N 个无序的整数中计算机最小的 K 个整数的算法，并给出时间复杂度，其中 K<<N，要求时间复杂度尽可能的低，不要求 K 个整数排序。
答：堆排序。将 N 个数中的前 K 个建立一个小顶堆。每读入一个新的整数，就把它插入到堆中，调整堆，但是每次调整都只调整前 K 个元素。从第 K+1 个位置开始的元素都忽略。时间为 NlogK
6、假设一个有 8 个 1024 字页面的逻辑地址空间,映射到一个有 32 帧的物理内存结构中，逻辑地址有多少位？

答：13
逻辑地址 = 逻辑页号 + 页内偏移
逻辑页面数为 8，因此逻辑页号长度为 3，页面的大小为 1024，因此页面偏移的长度为 10.
如果求物理地址多少位，则是 15
解：因为页面数为 8=2^3，故需要 3 位二进制数表示。每页有 1024 个字节，1024=2^10，于是页内地址需要 10 位二进制数表示。32 个物理块，需要 5 位二进制数表示（32=2^5）。
（1）页的逻辑地址由页号和页内地址组成，所以需要 3+10=13 位二进制数表示。
（2）页的物理地址由块号和页内地址的拼接，所以需要 5+10=15 位二进制数表示。
7、关于网络 ISO 各层协议的问题，把左右相对应。
应用层

网卡
表示层

路由 IP

会话层

交换机
网络层

TCP/UDP
传输层

HTTP/DNS
数据链路层

ASCII
物理层

PRC，SQL
答：貌似连线题？
（1）网卡的作用就是把数据进行串并转换（串连数据是比特流形式的，存在与本计算机内部，而计算机与计算机之间是通过帧形式的数据来进行数据传输的），MAC 子层规定了如何在物理线路上传输的 frame,LLC 的作用是识别不同协议类型然后进行 encapsulation(封包), 所以精确的说,网卡工作在数据链路层的 MAC 子层.
（2）路由 IP 属于网络层
（3）ISO 的术语称之为中继（relay）系统。根据中继系统所在的层次，可以有以下五种中继系统：
1.物理层（即常说的第一层、层 L1）中继系统，即转发器（repeater）。
2.数据链路层（即第二层，层 L2），即网桥或桥接器（bridge）。
3.网络层（第三层，层 L3）中继系统，即路由器（router）。
4.网桥和路由器的混合物桥路器（brouter）兼有网桥和路由器的功能。
5.在网络层以上的中继系统，即网关（gateway）.
我们经常说到的以太网交换机实际是一个基于网桥技术的多端口第二层网络设备，即数据链路层
（4）TCP/UDP 属于传输层
（5）HTTP/DNS 属于应用层
（6）表示层位于 OSI 分层结构的第六层，它的主要作用之一是为异种机通信提供一种公共语言，以便能进行互操作。这种类型的服务之所以需要，是因为不同的计算机体系结构使用的数据表示法不同。例如，IBM 主机使用 EBCDIC 编码，而大部分 PC 机使用的是 ASCII 码。在这种情况下，便需要会话层来完成这种转换。ASCII 属于表示层
（7）PRC，SQL 属于哪一层呢？
8、关于 Bridge 模式，Observer 模式，Strategy 模式，Mediator 模式，以上哪种模式可以使得算法的使用者忽视算法的具体实现？
答：Bride 模式
（1）Bridge 模式 的用意是"将抽象化(Abstraction)与实现化(Implementation)脱耦，使得二者可以独立地变化"。
（2）Observer 模式定义对象间的一对多的依赖关系,当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。
（3）Strategy 模式 定义一系列算法，把他们封装起来，并使他们可以互相替换。
将策略加以封装为一个物件，而不是将策略写死在某个类中，如此一来，策略可以独立于客户端，随时增加变化、增加或减少策略，即使是修改每个策略的内容，也不会对客户端程式造成影响。
（4）Mediator 模式 用一个中介对象来封装一系列关于对象交互行为。
9、数据库系统提供两种不同类型的语言，分别是自含式语言和嵌入式语言，来供数据库管理员及开发者管理，查询和更新。
10、数据库理论中取出右侧关系中所有与左侧关系的任一元组都不匹配的元组，用空值填充所有来自左侧关系的属性，再把产生的元组加到自然连接的结果上，这种连接运算称为？

答：左外连接
表的联结、运算符学习笔记
1.等值联结

两个表的相同列的值必须相等。
等值联结也称为 简单联结 或 内联结
2.非等值联结
非等值联结是包含非等号运算符的联结条件
3.外联结
通过外联结返回不直接匹配的记录。
外联结运算符只能出现在表达式的一侧，即缺少信息的那一侧。他将从一个表中返回在另一个表中没有直接匹配的行。
包含外联结的条件不能用 IN 运算符，也不能通过 OR 运算符链接到另一个条件。
4.自联结
自己联结自己的一种联结形式
5.交叉联结：
返回两个表的交叉乘积 这与两个表之间的笛卡尔乘积是相同的 CROSS JOIN
6.自然联结：
NATURAL JOIN 子句是以两个表中具有相同名称的所有列为基础。
它选择两个表中那些在所有匹配的列中值相等的行。
如果列具有相同的名称 但是数据类型不同，就会返回一个错误。
7.USING 子句
如果几个列具有相同的名称，但是数据类型不匹配，则可以使用 USING 子句来修改 NATURAL JOIN 子句 以指定要用于等值联结的列。
在多个列匹配时，使用 USING 子句只匹配一个列。
在引用列中不要使用表名或别名
对于使用 Using 限制只用一个相同列来关联的，where 条件当中出现的相同的列则必须限定为某一个表的列 否则因产生歧义而抛出错误
8.使用 ON 子句创建联结
自然联结的联结条件基本上是具有相同名称的所有列的等值联结。
要制定任意条件或指定要联结的列，可以使用 ON 子句。
联结条件与其他搜索条件分开。
9 INNER 与 OUTER 联结
在 SQL:99 标准中，只返回匹配行的两个表之间的联结叫做：内联结。
两个表之间的联结不但返回内联结结果而且返回左（或右）表不匹配行的结果。
两个表之间的联结不但返回内联结结果而且返回左联结和右联结不相匹配的结果，这样的联结就是完全外联结
关于左/右外联结的理解：
由于左右两个表完全匹配的情况称为 内联结，那么左外联结则可以理解为除了匹配的结果外，还将列出左表匹配以外的记录。
右外联结则是除了显示两表匹配的结果，还将显示右表除匹配结果以外的记录。
11. 下列关于索引创建的一般性原则，错误的是（）
A. 在经常用作连接的列上创建索引
B. 在经常用作排序的列上创建索引
C. 在经常搜索的列上及 where 子句的列上创建索引
D. 在定义为 text，image 和 bit 数据类型的列上创建索引
E. 在根据范围搜索的列上创建索引
12、关于数据库事务，什么是事务？简述事务的几个基本特性。并由低到高写出事务的几个隔离级别。
分析：数据库事务 ACID 特性：原子性、一致性、隔离性、持久性。。
隔离级别：未授权读取、授权读取、可重复读取、序列化
第二部分：C/C++必做题

1.哈希表的实现方式有哪几种，实现一种 hash_insert
答：直接寻址法、平方取中法、数字分析法、折叠法、随机数法、除留余数法。。
2.写一个程序，打印出以下的序列。
(a),(b),(c),(d),(e).....(z)

(a,b),(a,c),(a,d),(a,e).....(a,z),(b,c),(b,d).....(b,z),(c,d).....(y,z)
(a,b,c),(a,b,d)....(a,b,z),(a,c,d)....(x,y,z)
....
(a,b,c,d,.....x,y,z)

3.给出示例代码，如何限制一个类只在堆上分配和栈上分配
之前转载的那篇《C++内存管理》最后一章节——《对象内存大会战》有详细地描述。我转录如下：
禁止产生堆对象：
那么怎样禁止产生堆对象了？我们已经知道，产生堆对象的唯一方法是使用 new 操作，如果我们禁止使用 new 不就行了么。再进一步，new 操作执行时会调用 operator new，而 operator new 是可以重载的。方法有了，就是使 new operator 为 private，为了对称，最好将 operator delete 也重载为 private。现在，你也许又有疑问了,难道创建栈对象不需要调用 new 吗？是的，不需要，因为创建栈对象不需要搜索内存，而是直接调整堆栈指针，将对象压栈，而 operator new 的主要任务是搜索合适的堆内存，为堆对象分配空间，
禁止产生栈对象：
前面已经提到了，创建栈对象时会移动栈顶指针以"挪出"适当大小的空间，然后在这个空间上直接调用对应的构造函数以形成一个栈对象，而当函数返回时，会调用其析构函数释放这个对象，然后再调整栈顶指针收回那块栈内存。在这个过程中是不需要 operator new/delete 操作的，所以将 operator new/delete 设置为 private 不能达到目的。当然从上面的叙述中，你也许已经想到了：将构造函数或析构函数设为私有的，这样系统就不能调用构造/析构函数了，当然就不能在栈中生成对象了。
4.大概是下面这个样子吧，但愿我没把函数调用的地方记错。。。

```
#include <iostream>using namespace std;class A
{public:
    virtual void Fun(int number = 10)
    {
        std::cout << "A::Fun with number " << number<<endl;
    }
};class B: public A
{public:
    virtual void Fun(int number = 20)
    {
        std::cout << "B::Fun with number " << number<<endl;
    }
};int main()
{
    B b;
    A &a = b;
    a.Fun();
    system("pause"); return 0;  //虚函数动态绑定：B，缺省实参是编译时确定的。。。为 10
}
```

打印结果：B::Fun with number 10

```
#include <iostream>using namespace std;class A
{public:
    A(int j):i(j)
    {
        fun1();
    }
```

```
    ~A()
    {
    }

    virtual void fun2()
    {
        i++;
    }
    void fun1()
    {
        i *= 10;
    }

    int i;
};class B:public A
{public:
    B(int j):A(j)
    {
        fun2();
    }
    ~B()
    {
    }

    void fun2()
    {
        i += 2;
    }
    void fun1()
    {
        i *= 100;
    }
};int main()
{
    A* p = new B(1);
    cout<<p->i<<endl;
    delete p;
    system("pause");
};
```

打印结果：12

5 改错如下：

```
#include <iostream>using namespace std;class A
{public:
    A();
    ~A();

    int i = 0;
    static int j = 0;
```

```
    const int k = 0;
    const static char *p = "Hello world";
    static void fun();
};

A::A()
{

}

A::~A()
{

}static void fun()
{

}
```

10.3 是 C++各种成员变量的初始化问题。
主要是 staitc,const,static const 的问题；这里有详细地解答
http://blog.csdn.net/yjkwf/article/details/6067267
在 C++中，static 静态成员变量不能在类的内部初始化。在类的内部只是声明，定义必须在类定义体的外部，通常在类的实现文件中初始化，如：double Account::Rate=2.25;static 关键字只能用于类定义体内部的声明中，定义时不能标示为 static

在 C++中，const 成员变量也不能在类定义处初始化，只能通过构造函数初始化列表进行，并且必须有构造函数。

const 数据成员 只在某个对象生存期内是常量，而对于整个类而言却是可变的。因为类可以创建多个对象，不同的对象其 const 数据成员的值可以不同。所以不能在类的声明中初始化 const 数据成员，因为类的对象没被创建时，编译器不知道 const 数据成员的值是什么。

const 数据成员的初始化只能在类的构造函数的初始化列表中进行。要想建立在整个类中都恒定的常量，应该用类中的枚举常量来实现，或者 static cosnt。

```
class Test
{public:
    Test():a(0){}
    enum {size1=100,size2=200};private:
    const int a;//只能在构造函数初始化列表中初始化
      static int b;//在类的实现文件中定义并初始化
     const static int c;//与  static const int c;相同。};int Test::b=0;//static 成员变量不能在构造函数初始化列表中初始化，因为它不属于某个对象。cosnt int Test::c=0;//注意：给静态成员变量赋值时，不需要加 static 修饰符。但要加 cosnt
```

编程题：
编程题。编写代码把 16 进制表示的串转换为 3 进制表示的串。例如 x=”5”，则返回：”12”；又例如：x=”F”，则返回”120”

2010 网易校园招聘笔试题

一、填空
1、多任务系统里面，一个任务可以在占有资源的同时申请资源，这会导致_死锁_.
2、实现内联函数的关键词是_inline_
3、ping、tracert 是属于 tcp/ip 协议族里面的哪个协议？ ICMP
二、简答

1、请尽可能举出你所知道的数据库备份与还原的方法（数据库类型不限，只要知道的就写上）
备份：
1）直接拷贝要备份的数据库数据
2）oracle 用 rman 进行备份
3）用 sql 导入方式
4）日志备份
5）完整备份
6）差异备份
还原：
1）oracle 用 rman 进行还原
2）用 sql 导出方式
3）日志还原
4）差异还原
2 中断是什么？cpu 在中断的时候做了些什么？
答：中断就是中止当前正在执行的工作，而去执行引起中断的事件，当引起中断的事件执行完毕之后，CPU 继续执行以前的未执行完的工作。
CPU 暂时中断当前正在执行的程序而转去执行相应的时间处理程序
3、markfile 文件有什么作用
答：makefile 文件保存了编译器和连接器的参数选项,还表述了所有源文件之间的关系(源代码文件需要的特定的包含文件,可执行文件要求包含的目标文件模块及库等)
4、谈谈你对虚函数的认识，并写出实现虚函数的方法
答：虚函数的作用是实现动态联编，也就是在程序的运行阶段动态地选择合适的成员函数，在定义了虚函数后，可以在基类的派生类中对虚函数重新定义，在派生类中重新定义的函数应与虚函数具有相同的形参个数和形参类型。以实现统一的接口，不同定义过程。如果在派生类中没有对虚函数重新定义，则它继承其基类的虚函数。
虚函数实现方法：引入虚表
三、数学题：
1、1-9 这 9 个数字中，选 3 个出来，其和为奇数的组合有几个？
分析：分两种方式：3 个全是奇数或者 2 偶 1 奇，那么这种组合有：C(5,3)+C(4,2)*C(5,1)=10+6*5=40 种
2、请把 16 进制数 270f 转化为十进制数
分析：(270f)=2*16^3+7*16^2+15
附加：查看 tcp 连接的命令
1、netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'
例如输出：
SYN_RECV 1
CLOSE_WAIT 25
ESTABLISHED 122
FIN_WAIT1 1
FIN_WAIT2 12
TIME_WAIT 202
其中的 SYN_RECV 表示正在等待处理的请求数；ESTABLISHED 表示正常数据传输状态；TIME_WAIT 表示处理完毕，等待超时结束的请求数
CLOSED：无连接是活动的或正在进行
LISTEN：服务器在等待进入呼叫
SYN_RECV：一个连接请求已经到达，等待确认
SYN_SENT：应用已经开始，打开一个连接
ESTABLISHED：正常数据传输状态
FIN_WAIT1：应用说它已经完成
FIN_WAIT2：另一边已同意释放
ITMED_WAIT：等待所有分组死掉
CLOSING：两边同时尝试关闭

TIME_WAIT：另一边已初始化一个释放
LAST_ACK：等待所有分组死掉
2、lsof
lsof 命令的原始功能是列出打开的文件的进程
lsof -i :22 知道 22 端口现在运行什么程序
lsof 显示所有的进程
查看所属 root 用户进程所打开的文件类型为 txt 的文件：# lsof -a -u root -d txt

2011 网易校园招聘笔试题

1、写出输出：char array[] = “abcde”; char* s = array;
cout<<sizeof(array)<<strlen(array)<<sizeof(s)<<strlen(s);
答案：6545
2、什么是用户级线程和内核级线程？区别。
内核线程：线程切换由内核控制，切换的时候，要从用户态进入内核态，切换完毕要从内核态返回用户态；可以很好的利用 smp，即利用多核 cpu。windows 线程就是这样的。
用户级线程：用户态程序自己调度线程切换，不需要内核干涉，少了进出内核态的消耗，但不能很好的利用 smp。目前 linux pthread 大体是这么做的。
3、从 C++文件到生成 exe 文件经过哪三个步骤？
1）用户点击编译程序时，编译程序将 C++源代码转换成目标代码，目标代码通常由 机器指令和记录如何将程序加载到内存的信息组成。其后缀通常为.obj 或.o；
2）目标文件中存储的只是用户所编写的代码的转换结果，并不包括底层的操作指令，不能直接运行。例如程序包 iostream 实现了所有有关输入和输出的操作，并且其所有实现操作的机器代码都放在一个库中，库是对已实现的程序经编译后所产生的代码集合，用户可以在程序中直接使用库。
3）一个被称为链接程序的特殊程序将用户程序的目标文件和 iostream 库中必要代码链接起来生成一个可执行文件，其后缀通常为.exe 。这个可执行文件中包含了执行该用户程序所需要的所有机器代码，其过程大体如下所示：

4、有个二维数组 A(6*8)，每个元素占 6 字节，起始地址为 1000，请问最后一个元素 A[5][7]的起始地址为？？？ 数组 A 占内存大小为？？？ 假设以行优先，则 A[1][4]起始地址为？？？
答：A[5][7]起始地址：1000+(6*8-1)*6=1282
数组 A 占用内存：6*8*6=288 字节
A[1][4]起始地址：(1*8+4)*6+1000=1072
5、用 C 语言把双向链表中的两个结点交换位置，考虑各种边界问题。

```
Struct Node
{
    Node *prev;
    Node *next;
}
Void exchange(Node *node1,Node *node2)
{
}
```

```
void exchange（LinkedList p） {
    //交换 p 结点与其前驱结点的位置。
     q=p->llink; //q 是 p 的前驱结点；
     q->llink->rlink=p;
    p->llink=q->llink;
    q->rlink=p->rlink;
    q->llink=p;
```

```
    p->rlink->llink=q;
    p->rlink=q;
}
```

6、*.dll,*.lib,*.exe 文件分别是什么，有什么区别？
见：http://www.cnblogs.com/no7dw/archive/2010/11/23/1885890.html
用 C 语言把双向链表中的两个结点交换位置，考虑各种边界问题。
7、如右图所示，一个 n*m 的矩阵 M 中，标记 0 为白色区域，标记 1 为黑色区域，白色区域代表可以行走的区域，黑色区域代表阻挡，可以看到，如果在这个矩阵中只向上，下，左，右移动，那么有某些白色区域是不能到达的，我们称为这样的矩阵不是全相通的。
（1） 如何验证一个矩阵是不是全相通？请给出算法思路。
（2） 计算出你的算法的空间复杂度和时间复杂度
（3） 用 C/C++编写出代码，并在适当地方加上注释。

附加题（20）：使用八叉树算法把 24 位真彩色转化成 256 色。24 位真彩色包括 R,G,B 颜色，每种颜色 8 位。
有 11 盆花，围成一圈，要求每次组合时，每盆花相邻的两盆花与上次不同，请问有多少排列方法？
貌似是组合数学问题。。复习一下组合数学吧。。男士和女士跳舞的变形。。
集合问题，一个村有 70 人，进行 PVP 比赛，共有 4 个职业：骑士，牧师，法师，刺客。
已知有 34 人报名骑士，
24 人报名牧师，
13 人报名法师，
32 人报名刺客，
12 人既报名骑士又报名牧师，
13 人既报名牧师又报名法师，
……
3 人报名骑士，牧师和法师，
……
请问有多少人没有报名？以上数字为捏造。

```
Void foo(int source* src,int *dest, int N)
{
    Int count[256],index[256],i;
    For(I = 0;i<256;i++) count[i] = 0;
    For(I = 0;i<N;i++) count[source[i]] = count[source[i]]+1;
    Index[0] = 0;
    For(I = 1;i<256;i++) index[i] = index[ i-1] + count[i-1];
    For(I = 0;i<N;i++)
    {
        Dest[index[source[i]]] = source[i];
        Index[source[i]] = index[source[i]]+1;
    }
}
Int main()
{
    Int src[] = {3,5,3,6,10,8};
    Int dest[] = {0,0,0,0,0,0}
    Int N = 6;
    Foo(source,dest,N);
```

```
}
```

输出结果是什么？

改错题：

```
Char values[] = “NetEase”;
Int main()
{
    Char *buf;
    Int d = -1;
    Unsigned int ss = sizeof(values)/sizeof(value[0]);
    If(d<=ss)
    {
        Scanf(“%s %d”,buf,d);
        Printf(“%s,%s(%d)\n”,values,buf,d);
        If(d = 666)
            Printf(“you are lucky!”);
    }
    Return 0;
}
```

2 只宠物合成，1 只有 5 技能，1 只有 4 技能，每个技能有 a%概率遗传，请问刚好有 7 个技能遗传成功的概率是？

```
Public class A
{
    A(){cout<<”1”;}
    A(A &a){cout <<”2”;}
    ~A() {cout<<”3”;}
}
Public class B
{
    B(){cout <<”4”;}
    B(B &b){cout<<”5”;}
    ~B(){cout<<”6”;}
}
Int main()
{
    A* pa = B(A());
    Delete pa;
    Return 0;
}
```

输出结果是什么？

1.写一个函数，打印一个如下的 n x n 的矩阵
例如：

n = 5
1 1 1 1 1
1 2 3 2 1
1 3 6 3 1
1 2 3 2 1
1 1 1 1 1

n = 6
1 1 1 1 1 1
1 2 3 3 2 1
1 3 6 6 3 1
1 3 6 6 3 1
1 2 3 3 2 1
1 1 1 1 1 1

提示：除了边上的元素,每个元素都是由边上的某两个元素相加得到的

void AddMatrix(int n);

2.有一个人站在电影院门口卖票，票价 50，一开始手上没有找零的钱，
现在有两种人来买票，A 拿着 100 元的钱，人数为 m(m<20)，B 拿着 50 元的钱，
人数为 n(n<20)。卖票的人必须用从 B 类人中那里得来钱找给 A，所以卖票
的顺序是有限制的。
要求写一个程序打印出所有的买票序列：

例如：m =2 ,n = 3;

BABAB
BBAAB
BBBAA
BBABA

void Ticket(int m, int n);

以上两题可选的实现语言：c/c++/java