《数据结构与算法》常见问题解答

URL: http://db.pku.edu.cn/mzhang/ds/resource/FAQ.pdf

张　铭

修订于 2006 年 12 月 4 日

目 录

一、教材信息

1. 教材出版信息

（1） 主教材： 许卓群、杨冬青、唐世渭、张铭，《数据结构与算法》，高等教育出版社，2004 年 7 月。主教材勘误表。
（网上购书：http://www.landraco.com.cn/book/bookdetail.asp?plucode=14616-00）

（2） 辅助教材：张铭、赵海燕、王腾蛟，《数据结构与算法——学习指导与习题解析》，高等教育出版社， 2005 年 10 月。ISBN 7-04-017829-X。辅助教材勘误表。
（网上购书：http://www.landraco.com.cn/book/bookdetail.asp?plucode=17829-00）

（3） 主教材和辅助教材都在北大教材科、王府井图书大厦、西单图书大厦有售。某些新华书店也可能有售。
高教社购书热线：010-58581118，58581117，58581116。
高教社网上订购 URL：http://www.landraco.com.cn

2. 每位作者负责的章节

主教材第 1、2、3 章（第 1-84 页）由许卓群主笔，第 4、5、6 章（第 85-202 页）由杨冬青主笔，第 8、10 章（第 254-293 页，第 333-358 页）由唐世渭主笔，第 7、9、11、12 章（第 203-253 页，第 294-332 页，第 359-468 页）由张铭主笔。

辅助教材第 7、9、10、12、13、14 章（第 162-215、244-279、280-303、324-503 页）由张铭教授主笔，第 1、2、3、11 章（第 1-47、304-323 页）由赵海燕副教授主笔，第 4、5、6、8 章（第 48-161、226-243 页）由王腾蛟副教授主笔。

3. 课程网站（课程讲义、算法源代码等）

http://www.db.pku.edu.cn/mzhang/DS/

4. 关于教材内容问题（包括错误和疑问）等数据结构技术性问题

请到 http://db.pku.edu.cn/mzhang/DS/bbs/index.asp 论坛注册账号，参与讨论。

请不要给作者发 email 询问教材问题，作者非常忙，很可能没有时间直接回答您的问题。我们 BBS 上面有助教负责整理错误和问题，并且尽量解答，而且及时更新到本勘误表中。

二、 数据结构的一般问题

1. 伪代码到底是什么？

答：http://algorithm.myrice.com/algorithm/pseudocode.htm 给出伪代码定义：伪代码(Pseudocode)是一种算法描述语言。使用为代码的目的是为了使被描述的算法可以容易地以任何一种编程语言(Pascal, C/C++, Java, etc)实现。因此，伪代码必须结构清晰，代码简单，可读性好，并且类似自然语言。

我们教材所用到伪代码，就是半描述半代码性质的，可以说是 C++伪代码。某些代码细节可以忽略，甚至有些实现细节可以屏蔽掉。

当然在作业里，关键性算法的代码还是要有的。实在写不出 C++伪代码，可以写出像教材第 38 页中缀转后缀算法（注意第一次印刷版该算法有错，勘误表已经纠正）那样的类自然语言代码。

2. 什么是“程序的鲁棒性”？

答：鲁棒性，是英文“Robust”的音译，指程序的健壮性。就是程序对异常情况的处理能力，例如不合理的输入、下标越界等。鲁棒性差的程序在遇到异常情况时会造出内存泄漏或是死循环等情况，而鲁棒性强的程序就能很好的处理这些异常。

3. 教材中多处提到 log n 是以 10 为底的对数吗？

答：数据结构和算法类的教材中，大多数是 log n 是以 2 为底的对数。其实以 2 为底和以 10 为底的对数从算法复杂性来看是等价的。

$\log_a n = \log_b n / \log_b a$ ，对于变量 n 和任意两个整数变量 a 和 b，$\log_a n$ 与 $\log_b n$ 只相差常数因子 $\log_b a$，而与 n 的值无关。教材中的大多数代价分析都忽略常数因子，因此有关对数的复杂性分析与对数的底数无关。

4. 请问效率分析是只要记住结果还是要会自己推导？

答: 两个层面：记住最基本的算法的时间/空间代价和会推导一些不算太复杂的算法效率。

各种算法大体的的时间代价需要记住，算法的最好最差时间/空间效率一般应该会分析。那些特别复杂的时间代价推导，例如用到快速排序的递归推导是不会要求的。

一些不算太复杂的算法效率分析，肯定需要理解和自己会推导。比如给你一个不太难的算法，请你估计复杂度（即时间代价）；或者对你自己写的算法，分析其时间代价。

5. 关于算法

算法语言无所谓，只要能看懂。考试用 C++出题，但答题随意(可以用 C/C++、Java、Pascal、自然语言等等，看得懂就可以)。

如果要求自己独立地写算法（而不是填空），请注意写算法思想，并加上足够的注释。

在平时作业中，对于算法中直接使用的类和函数（例如栈、队列的函数），最好先写 ADT，并说明函数功能、入口参数、出口参数。尤其是同学们自己定义的类和函数。

6. 怎样写抽象数据类型 ADT？

抽象数据类型是定义了一组运算的数学模型。它与物理存储结构无关，使软件系统建立在数据之上。从这个意义上来说，当我们定义一个抽象数据类型时，必须说清楚两点：其一，它的逻辑结构；其二，定义在这个逻辑结构之上的抽象函数。

从抽象数据结构的观点来看，对数据结构的一切访问都是通过函数来进行的。不主张把属性直接 public，让外界直接访问。

请同学们不要拘泥于抽象数据类型的具体形式，它只是描述数据结构的一种理论工具。不一定要用 Interface 或者抽象类来表示。

我们提倡以下的定义方法：借用 C++语言的类定义形式，在定义抽象数据类型时，先用注释的形式说明其逻辑定义，然后定义其主要的运算函数（够用就行了，并非越多越好），并以注释的形式对函数功能加以说明。

7. 关于编程经验问题

答：编程上要靠自己平时多练习了，每次把自己的作业和推荐作业比较一下，看看差在什么地方。思路同学们都差不多，差在经验上。包括教材上的例题也可以在 VC 上自己实现看看，这些都是非常经典的算法，从算法的思想到代码结构都是值得借鉴的。

8. 不同小组的作业难度不同，怎么算成绩？

问：如果仅仅是简单相加的话，我觉得这样有失公平。每个组的题目不一样，难度也不一样。相对来讲，做难题的那一组同学肯定吃亏。还望助教老师想出一个平衡的方法，并在网上公布，以解除大家的疑惑。

答：助教在给分的时候会考虑题目的难易程度，况且平时作业的评分都比较高，只要你按时交了作业并且态度端正，就可以得到相对满意的成绩。请大家不要担心。

数据结构与算法课程比较重视平时的作业和上机练习。大多数情况下，我们按照“平时(书面作业)20%+上机(+报告)15%+期中 20%+期末 40%+考勤和态度 5%”给出成绩。不过，如果期中、期末两次考试都没有上 50 的学生，期末绝对不能及格。考试还是一个很过硬的指标。

例如，学生两次考试都得 70 分，每次作业得 8.5，假设态度分也是 5×0.85，那么期评成绩就是 76。总分可以提高一大截。

三、线性表和串疑难解答

9. 教材第 1 版第 1 次印刷 p19 倒数 L1，P21，P62 等处的“算子”一词，本身是纯数学概念，C++中似乎不应如此解释。

答：同意。其实，“算子”在英语中对应的是“function”，在中文的不同环境中应该翻译为“函数”或者“运算符”。

例如，P21 的 list ADT 代码，“算子”应该修改为“函数”。其他地方的“算子”，

(1) 如果是英文命名的，则称“函数”；

(2) 如果是“＋”、“=”、“>”、“<”、“>>”、“<<”等，应该称“运算符”。

第 4 次印刷版即将改正。

10.p24 算法 2.1，插入之前 curr 的值不确定？

答：限于篇幅，没有给出某些成员函数的具体实现。list 构造函数的初始化应该为：

```
template <class ELEM>
list::list(const int size)                    // 构造函数，初始化
  { msize = size;   curr_len= curr = 0; nodelist = new ELEM[size]; }
```

11.assert 断言的使用问题。

答：断言 assert 能帮助在程序中发现逻辑错误，它仅在 Debug 版本起作用，用于检查“不应该”发生的情况。

边界判断是算法的重要组成部分。本教材中使用的 assert，并没有损坏算法的完整和正确性。

在实际系统中，读者可以根据需要处理某些异常。例如，把 P24 算法 2.1 和 2.2 中的 assert 置换成相应的异常处理代码，或者在调用算法 2.1 的 insert 或 2.2 的 remove 函数前进行异常判断并作出相应处理。”

12.教材第 26 页链表部分 first 指针的含义？

问：教材上 first->link 才指向第一个结点。first 指向的首结点不算是数据元素，那这个首结点有什么意义？这与我们通常的编程习惯不同。请问考试时是否默认采用教材中的规定？

答：教材上采用的是带头结点（header node，也称“头结点”，即链表首部的虚结点）的链表，头结点的 data 域用来存储整个链表的简单信息。因此寻找第 i 个结点的算法 2.3

FindIndex(i) 中，变量 first 的指向的是一个特殊的首结点。first->link 才是。

算法 2.3 假定在进入算法前查找链表中第 i 个结点时， i 按照 C/C++数组下标编号的规则，是从 0 到 n-1。链表的第 0 个结点应该是 first->link，即链表真正的第一个有效结点。加入头结点后，可以使得很多操作的边界处理变得比较简单。例如，在空链表中插入元素，在链表首部插入元素，删除后链表成为空表，等等。

13.什么叫循环队列？如何区分空满循环队列？

答：所谓循环队列是不是就是教材中第 52 页的顺序队列。一般 front 指向队首第一个元素，rear 指向下一个该插入信息的位置。有些教材用 rear 指向循环队列最后一个元素，而且链式队列的 rear 往往指向队列的最后一个有效元素（例如教材第 53 页算法）。

如果循环队列带上了当前的长度信息，那么可以利用所有的结点空间，如教材 51 页算法所示。

如果没有带长度信息，如果不加 int curr_len 记录队列长，那么需要利用 rear 和 front 的关系来判断，需要牺牲一个结点的空间。整个队列还有一个空位置“(rear+1) % maxsize == front”时，就要报告“队列满”。rear=front 表示为空。

14.关于递归转化

问：Hanoi 塔问题，也可以用迭代来解决的，我看过迭代的解法，甚至在我的一本参考书上，就写着：……any recursive problem can also be resolved iteratally。而在您翻译的《数据结构与算法分析——C++版》上，关于 Hanoi 塔的问题，写的是：并不是所有的递归问题都可以用迭代来解决……Hanoi 塔问题就不能用迭代来解决。我感兴趣的是，Hanoi 塔问题的确是可以用迭代解决的，但是是否所有的递归问题都可以用迭代解决，好像需要证明。我只是直觉上感觉可以从递归问题的出口反过来推导解决递归问题，不知道有没有进一步的证明，您能不能给一点指导？

答：张铭翻译的《数据结构与算法分析——C++版》（电子社 2002 年版）第 81 页，"并不是所有的递归问题都可以用迭代来解决"，这里的 iteration 是指例 4.2 那种简单循环递推。在计算机术语中，我们所指“迭代”、“递推”都是类似含义，你的参考书比较特别。

递归问题都可以用栈来化解为非递归的形式。如果是类似于二叉树访问地对问题空间进行搜索，则可以采用教材第 94-97 页，算法 4.4-4.6 的那三个非递归深度优先周游框架。例如，Hanoi 塔问题可以采用中序非递归深度优先周游框架来消除递归（注意修改进出栈参数，修改算法中的 Visit 语句）。

对于图 2.15 的栈图示。建议：

（1）应该增加一个返回地址位，表示是从第一个递归语句深入还是从第二个递归语句深入。就像二叉树后序周游那样。

（2）类似于二叉树中序周游，简化 Hanoi 进栈的内容。从第一个递归语句返回进入第二个递归语句时，本层内容不需要压栈。

推荐参考讲义：张铭 2002 年网站 http://www.db.pku.edu.cn/mzhang/site/03_recrule.pdf。

推荐参考书：严蔚敏等《数据结构》 或者许卓群等《数据结构》1985 年版。

15.关于双端队列和双栈的区别

答：双栈是一种加限制的双端队列，它规定从 end1 插入的元素只能从 end1 端删除，而从 end2 插入的元素只能从 end2 端删除。

另外，栈和队列都是特殊的双端队列，对存取加了限制。在大部分的插入和删除运算发生在队列的一端时，可以用双端队列提高效率，这时它的时间复杂度为常量级。

16.关于 String 类

(1) <stdlib.h>是干什么的？

答：<stdlib.h>和<string.h>类似，是 C 语言的一个 library,包含一些常量的声明和常用的函数，比如 rand()函数（产生随机数）等。

(2) #include <string.h>后，声明字符串变量 “string s1”会报错 undeclared identifier？但是其中的一些函数如 strlen 却可以使用？难道要自己编写 string 类？

答：(a) 如果没有编写自己的 string 类，#include <string.h>之后，不能直接声明 string s1。因为<string.h>是系统的 library，里面并没有定义 string 类，只是定义了一些 char*字符串的常用函数（其中也包括 strlen()），书上有自定义的 string 类，可以敲出来，并#include "string.h"（注意，是“”而不是<>）。

(b)事实上，可以直接引用 STL 中定义的标准 string 类。首先要“#include <string>”，注意，没有“.h”。然后，需要

using namespace std;

或者声明 using std::string;

这样，定义 string s1 就不会出错了。不过请注意，教材第 62 页自己定义的 Sting 类与 STL 的 string 类略有差别。

(c) 另外，教材上定义的 strlen()是 String 类的 Public 函数，必须针对一个 String 对象引用，单独引用 strlen()则是调用标准 C 的函数。

String P; int len

len = strlen(P); 或者 len = P.strlen();

(3) strlen()和 length()

问：我看了您的勘误，感觉有个小问题：就是 String 类型的变量本身提供了 length()的 member function，而勘误上面似乎写的还是 strlen()，我编写程序和学习 C++时用到的、看到的都是用 length()，所以我觉得应该用 length()更为恰当一些。

答：标准 C++类(STL)确实定义了一个 length()。不过，教材第 62-63 页自己定义了一个 String 类。函数名称只是代表，叫什么都无所谓。教材第 63 页第 5 行定义了 strlen()，因此我们沿用 strlen()。

(4) string&中“&”是什么意思啊？

答：string&的&表示引用，返回的是一个引用值，所谓引用指的是对一个对象的引用。如果一个函数返回的是一个引用类型，那么该函数可以被当作左值使用。如果一个对象或表达式可以放在赋值号的左边，那么这个对象和表达式就叫左值。

17.教材为什么要自己定义一个 String 类？

答： 教材定义 String 类是为了详细解释串的实现细节、运算符重载等。我们在讲解新的数据结构时，需要了解那些标准数据结构及其函数底层的实现原理。

我们提倡采用 string.h 中的标准函数 C++的<string.h>函数库、STL 的<string>类、VC 的 String 类。

类似地，教材定义了栈、队列等基础数据结构

任何时候，我们都提倡使用 STL，除非有特殊训练需要学生自己设计。

18.局部对象在语句块结束执行后将被释放。算法 3.9 能 return temp 吗？

答：语句显然不符合逻辑。temp 是语句块内的局部对象，语句块结束执行后将被释放，返回其结果（return 语句并不能返回对象的值，它只能返回一个标准表达式的值）将导致系统错误（在 C++中实际测一下即可，我已经实际测试过了）。此算法需要在退出前重新分配 str 空间，然后将 temp.str 复制回 str；

函数体执行完后，会是释放局部变量，但是不会释放由这些局部变量申请出来的动态空间。return 语句完全可以返回对象，包括（临时变量）对象的内存空间和值。

问题出在析构函数~String( )在销毁对象时释放了空间。因此退出函数体时，temp 申请的空间被释放了。

解决方案只需要把析构函数~String( )的“delete [] str;”语句删除掉就可以了。本教材在第 4 次印刷时修改了 P74 算法 3.8。

【算法 3.8】

```
// 析构函数 destructor
String::~String(void) {
//   delete [] str;
}
```

【算法 3.8 结束】

析构函数~String( )什么也不做。String 类的字符空间应该由程序员主动 delete 释放。因为拼接运算符要求返回 String 型的值，相关函数中的临时变量申请的空间必须返回。例如 算法 3.9 中临时变量 temp 的对象空间必须返回到上一级。如果在析构函数中销毁了 temp 的 str 空间，则调用算法 3.9 时将出现错误。

19.第 3.3 和 3.4 节的代码可以简洁明一些

算法 3.2

```
int strcmp(char *d, char *s) {
    int i;
    for (i=0;d[i]==s[i];++i ) {
```

```
            if (d[i]=='\0' )        //  此时 s[i]==d[i]
                return 0;           //  两个字符串相等
        }
        return (d[i]-s[i])/abs(d[i]-s[i]);   //不等,比较第一个不同的字符
}
```

算法 3.14

```
int FindPat(String S, String P, int startindex) {
        //g 为 S 的游标，用模板 P 和 S 第 g 位置子串比较，
        //若失败则继续循环
        for (int g= startindex; g <= S.strlen() - P.strlen(); g++) {
            for (int j=0;   ((j<P.strlen()) && (S[g+j]==P[j])) ; j++)
                ;
            if(j == P.strlen())
                return   g;
        }
        //若 for 循环结束，或者 startindex 值过大，则整个匹配失败，返回值为负，
        return(-1);
}
```

20.KMP 算法疑问

问：在算特征向量的时候，按 04 年版新教材的全都是非负整数，我感觉没有－1 的话在匹配时模式右移好像有问题啊。比如说当模式的第 0 个字符比较时不等，模式应该右移多少位？

答：第 0 个字符不匹配的时候，模式指针不右移，仍指向 j=0；指向字符串的指针向 i 右移一位，根本就没跳入模版对准的循环里。

21.关于线性表和串的作业题意

(1) 第 55 页 2.4 题解意

答：第一题大部分同学认为 F 点是孤立的，所以没有对 F 进行操作。需要先把 f 点剥离出来，然后再作 append 操作。

(2) 书上习题 2.9 中 top 运算是什么运算，是否读取栈顶内容

答：等价于 p34 的 gettop 操作,取出栈顶元素但不 pop。

(3) 2.17 题那个迷宫题是什么意思？

问：是不是如果 maze(j,k)=0 的话，就可以检查 maze(j-1,k)、maze(j-1,k-1)、maze(j-1,k+1)、maze(j,k-1)、maze(j,k+1)、maze(j+1,k)、maze(j+1,k+1)、maze(j+1,k-1)这八个值是不是 0，如果是 0 就可以继续走下去？

答：题目也没说是一个格子是可以向周围四个格子走，还是可以和 8 个方向都可以走。大家先按只能向周围 4 个方向走吧。走迷宫，就是搜，不用用到什么复杂的数据结构。当然你可以把他看成是三叉树，搜索的本质也就是树的遍历。

(4) 第 55 页 2.18 题解意

答：其实还是在一个队列，逻辑上分为三个子队列，具有三种优先级。一个时刻，出队操作只能在其中一个子队列中进行。事实上，总是都按照“高、中、低’的顺序来处理这三个队列，较最优先级没有处理完，不会去处理较低的那一个。

(5) 3.1 String A = " "//一堆空格也不知道是几个，要求 A.strlength()。

答：应该是印刷的问题，习题集 P40 页的解答按照没有空格算。

(6) 3.1 FindLast 函数在书中没有定义，怎么处理？

答：FindLast 在 66 页。

(7) 那个上机题 3.1 是不是要用类编啊？还是用个字符数组删除掉中间字符就行了？

答：因为题目中没有限制实现的方法，所以你可以采用你认为最好的方法。数组，或者其他的方法，可以比较多种方法在时间和空间上的好坏。

(8) 3.3 题目中字符串 S 与 P 中的字符间有没有空格？

答：应该是没有。

三、二叉树和树问题

22.关于满二叉树的定义

问：我发现其他教材上的对于满二叉树有如下定义：

(a) “一棵二叉树，如果所有分支结点都存在左子树和右子树，并且所有叶子结点在同一层上”

(b) “深度为 k 的二叉树，满足结点数目为 2 的 k 次方减 1”

在您的教材里，一棵二叉树，结点度为 0 或者 2，（或者为叶子结点，或者分支结点都存在左子树和右子树）。

请问怎么解释？

答：这是由于满二叉树、完全二叉树的不同定义引起的误解。

有些教材与我们正好相反，把两个定义完全倒过来了，例如你总结的定义(a)。有些教材中完全二叉树定义与我们一致，而满二叉树定义不同，例如你总结的定义(b)。

采用不同的教材时，请注意遵循其定义体系。例如在我们教材的满二叉树定义下，第 88 页给出了两个重要的性质：满二叉树定理和满二叉树定理推论。

我们的定义与国外主流教材一致。不妨这样来记忆我们教材中的定义：完全二叉树一般比满二叉树要宽，因为完全二叉树的每一层都尽可能地宽。

23.关于教材第 96 页的二叉树非递归后序周游的算法

问：我个人感觉教材中两处的 aStack.pop();和倒数第四行代码 aStack.push(element);都可以去掉，但是要在 Visit(pointer->value());前加一个 aStack.pop();我想这样的话，能减少进栈和出栈的次数，进而提高时间效率。

答：这样做是错误的，这个同学的初衷是希望减少对栈顶元素进行“弹出——修改——压入”操作，进而改成“直接修改栈顶元素对应的变量”操作，从而提高效率。但是，对应栈顶

元素的变量 element 在其修改期间可能会被覆盖。建议这位同学用一棵二叉树作为实例，用教材中对应的算法走一遍，就可以清楚地看到算法流程了。

24.教材第 115 页倒数第 9 行 GetParent()是私有的，不能这么调用吧？

答：没有问题，成员函数都可以调用私有函数和私有变量。

25.教材第 116 页中部的两段代码边界条件的讨论

问：当 pointer 的左孩子无右子树，下面的赋值不对吧？

```
if（tempparent==NULL)
    pointer->left=temppointer->leftchild()
```

答：教材上的算法是正确的。

temppointer 是用来替换待删除结点的，它是 pointer 的左子树中的最大值，tempparent 为 temppointer 的父结点。tempparent 为 NULL 时，就是 pointer 的左孩子无右子树的情况，也就是说 temppointer 是 pointer 的直接左孩子。此时，应该把 temppointer 的左孩子直接挂到 pointer->left，也就是把 temppointer 这一个结点脱链。

教材中算法 4.15 的代码有些冗余，我们可以简写为：

```
template <class T> void BinarySearchTree<T>::DeleteNodeEx
  (BinaryTreeNode<T>* pointer)
{
    if( pointer == NULL )
        return;              //若待删除结点不存在，返回

    BinaryTreeNode<T> * temppointer;          //保存替换结点
    BinaryTreeNode<T> * tempparent = NULL;      //保存替换结点的父结点
    //保存删除结点的父结点
    BinaryTreeNode<T> * parent = GetParent(root, pointer );
    if (pointer->leftchild() == NULL )   //如果待删结点的左子树空，就将它的右子树代替它
        temppointer = pointer->rightchild();
    else   {   //左子树不为空，在左子树中寻找最大结点替换待删除结点
        temppointer = pointer->leftchild();
        while (temppointer->rightchild() != NULL ) {
            tempparent = temppointer;
            temppointer = temppointer->rightchild();
        }   // end of while
        if (tempparent==NULL) //删除替换结点
            pointer->left=temppointer->leftchild();
        else tempparent->right=temppointer->leftchild();
        temppointer->left=pointer->leftchild();
```

```
            temppointer->right=pointer->rightchild();
        }   // end of else
        //用替换结点去替代真正的删除结点
        if (parent==NULL)
            root=temppointer;
        else if( parent->leftchild() == pointer )
            parent->left=temppointer;
        else parent->right=temppointer;
        delete pointer;
        pointer=NULL;
}
```

26.关于最大值堆和最小值堆的问题

问：对数组进行升序排序的时候用的是最大值堆还是最小值堆？如果用最小值堆，筛完以后数组是按降序排列的，是不是还要把它倒过来？

答：利用堆进行排序时，你可以自己选择采用最大值堆或最小值堆。如果用最小值堆，筛完以后，还只是建立了一个初始堆，要像教材第 233 页算法 7.16 那样，充分调用 n-1 次 RemoveMin()的操作，才能使得整个数组最后按降序排列的。是否还要这个降序数字倒置过来取决于你的应用需要。

另外，请注意一次简单的建堆操作并不保证数组的升序或者降序排列，也就是说堆的实现数组并不是一个全排序。堆的性质（教材第 116 页定义)只能保证它在完全二叉树表示中比子结点的值小（最小堆）或者大（最大堆）。

27.二叉树和树的计数问题

n 个结点的不同形态二叉树的数目 b_n，等价于{1, 2, ..., n} 顺序进栈、出栈所得序列数目，等价于前序序列为{a_1，a_2，.....,a_n}的二叉树数目，等价于 n 个结点的标号 bst 树数目，等价于 n+1 个结点的不同形态树（不是森林）的数目，其值为为 Catalan 函数：即：

$$b_n = C_{2n}^{n} - C_{2n}^{n-1} = \frac{1}{n+1} C_{2n}^{n}$$

28.教材第 139 页树的链式存储：指针、索引和下标。

问：在树的链式存储中，不止一次提到了索引，但讲得不清楚，一会儿说是指针，但一会儿又说以结点的下标表示。请问这里的索引究竟是指向结点的 T*型还是表示下标的 int 型？

答：指针和下标本质上都是索引，他们都是对于存放在某一个地址的数据的引用。其中，我们通常所说的“指针”往往指 new 出来的动态指针，而针对下标型的所谓“指针”针对数组中的某个偏移位置而言。

教材中用数组中的下标来实现的树的存储，是一种“静态”链式存储。

29.学习指导上 p96 习题 5.4

问：在学习指导上 p96,习题 5.4 的解答中，在方法三：非递归法解法时，在 while 循环里最外层的 else（while 里的第二个 else)里是否少了一种条件？如果 pointer1 和 pointer2 一个为空，一个不空（即一个树有子树，但另一个没有），和当 pointer1 和 pointer2 同时为空（既两棵树均没有子树时）时都将将进入最外层的 else 里，这似乎不正确。

答：习题集是正确的。应该参考 while 循环的条件，此处的 else 表示满足!aStack.empty()&&!bStack.empty()，而不是你说的那种可能，你说的情况不满足 while 循环条件。

四、图的问题

30.关于教材中第 146 页算法 5.7 的 PrevSibling 函数中 while (pointer)循环

问：第 17 行第一个 while (pointer)循环与 while(!aQueue.empty)中的第二个 while (pointer)循环是什么关系？两者能不能合在一起写？

答：第一个 while (pointer)循环为了先向队列中压入一些结点（根结点以及其右兄弟结点），因为后面需要进行队列为空的判断，这是前期处理。其代码功能与 while(!aQueue.empty)中的 while (pointer)是一样的。

两者可以合并为：

```
template <class T>
TreeNode<T>* Tree<T>::PrevSibling(TreeNode<T>* current)
{   //返回 current 结点的前一个邻居结点
        using std::queue;                                   //使用 STL 队列
        queue<TreeNode<T>*> aQueue;
        TreeNode<T>* pointer=root;                          //标识当前结点
        TreeNode<T>* prev=NULL;                             //标识当前结点的前一个兄弟结点
        //当前结点为空、树为空或所求结点为根结点时，返回 NULL
        if ((current==NULL)||(pointer==NULL)||(current==pointer))
                return NULL;
        while (pointer != NULL || !aQueue.empty())    {
                if (pointer)    {
                        if(pointer==current)
                                return prev;
                        aQueue.push(pointer);
                        prev=pointer;
                        pointer=pointer->pSibling;          //沿当前结点右兄弟结点链寻找
                } //end of if
```

```
            else {
                prev=NULL;
                pointer=aQueue.front();
                aQueue.pop();                    //出队列
                pointer=pointer->LeftMostChild();    //下降到左子结点
            }
        }//end while
        return NULL;
}
```

31.教材第 181 页中的 DFS 中写了 PreVisit 和 PostVisit

问: PreVisit 和 PostVisit 各有什么用？其中有一个不就可以了吗？为什么写两个？

答：前者是先访问当前结点，再访问相邻结点；后者是先访问相邻结点，再访问当前结点。如果图是一个有向树的话，PreVisit 就是先根周游，PostVisit 就是后根周游。可以根据实际需要把这两个函数实现为具体的操作，比如打印当前结点的编号，也可以是个空函数。大多数情况下，我们是在 PreVisit 的地方进行处理。不过，有时候可能需要用到 PostVisit 进行后处理。例如，教材第 186 页 Do_topsort ( )中，实质上就是用到了后访问 PostVisit。

32.Dijkstra 算法怎么保证从第二组所有的顶点中选取到源点距离最小？

问：每次是从第二组所有的顶点中选取到源点距离最小的，但是那个最小堆并不包括所有的第二组结点，那么我们何以保证它的最小性？

答：第二组的其他顶点目前到 s 的距离都为无穷大，根本没有必要加入到堆中。当找到最小值顶点后，第二组的相关顶点到 s 的距离可能会被刷新。教材第 190 页算法代码的最后一个循环进行刷新处理。那些到 s 的距离值变成一个确定值（不是无穷大）的顶点，将会被插入堆中；到 s 的距离变小的顶点，都将会被再次插入堆中。以前的值还在堆中，但现在插入的比原先的小，根据堆的性质，不必担心会取到那个过时了的更大值。

33.第 190 页倒数第八行应该是多余的

因为如果是 visited 过的就肯定是最小的了，不可能出现这个情况。

```
//刷新 D 中的值，因为 v 的加入，D 的值需要改变，只要刷新与 v 相邻的点的值
for(Edge e=G.FirstEdge(v); G.IsEdge(e);e=G.NextEdge(e))
    if(D[G.ToVertex(e)].length>(D[v].length+G.Weight(e)))
      // &&G.Mark[G.ToVertex(e)]== UNVISITED 冗余
    {
        D[G.ToVertex(e)].length=D[v].length+G.Weight(e);
```

```
            D[G.ToVertex(e)].pre=v;
            H.Insert(D[G.ToVertex(e)]);
        }
```

34.图的题意

(1) 请问习题 6.10 是什么意思啊？？？严重的不明白。

答：可将 6.10 中的“最大可能下限”简单地理解为“下限（即算法时间的下限）”。

(2) 习题 6.15 环和连通性的问题。

6.15 中提到了有向连通图的概念，而我们在离散数学中学到的是：有向图的连通分为弱连通、单向连通、强连通这三种情况，没有单独的连通这个概念，而这本教材上也没有给出定义，请老师给出详细的定义，不然作业就没法做了，谢谢。

答：对环的定义，对有向图的环，路径为 2 即可成环。数据结构的所谓“连通”，缺省情况指“弱连通”。习题 6.1 中中的无环有向连通图指一般的连通概念，即任意两个顶点间有路径可达，并不要求互达。

(3) 习题 6.19 是求一个强连通分量吗？

答：题目中的“有向图”应该是“有向无环图”，且题目要求找出所有的强连通分量。

(4) 习题 6.20 是什么意思？好像书上都实现了啊。那这道题是要我们做什么与书上实现得不一样的事？

答：写一个算法就可，该算法不一定要与书上的实现不一样。

(5) 习题 6.21 权值为负的路径是什么意思？一个图的权值为负，那么怎么定义它的路径长度？是应该找权值最大的，还是最小的路径？有点不懂

答：一条路径长度等于路径上所有边的长度之和，如果是负的也累加就是了。还是找权值最小的路径。我们一般考虑的都是权值为非负整数的情况。但这道题假设权值为负数的情况，并由此问了一些问题，让我们解答——题意如此。

(6) 上机题 6.1 是否可以设定 A[i,j]*A[j,i] = 1

答：可以，这并不影响算法的设计。在现实中，由于银行的买入卖出价是不一样的（银行要赚钱，还不包括手续费），A[i,j]*A[j,i] < 1。

五、排序和检索问题

35.关于教材第 218 页的算法 7.11 Shell 排序的增量

(1) 问：如果增量不是 2 的整数次幂，比如 7 个数据，那么增量 delta 会变成 7/2 = 3, 3/2 = 1。这是合理的吗？如果合理，那么增量序列岂不是变成了{3,1}。也就是说写出一个 shell 算法以后，对于不同的 n，增量序列有可能是不一样的？

答：这是合理的。对于 n 为 7 的情况，经过 2 轮排序就可以了。

(2) 问：Shell 排序的代码下标传递正确吗？以课本上图 7.4 作为例子 n=8，Array[]={45,34,78,12,34,32,29,64} 。当 delta=4 时，主函数 sort() 里的“ModifiedInsertsort(&Array[j],n-j,delta);”语句所传的参数依次为(&Array[0],8,4)，

(&Array[1],7,4)，(&Array[2],6,4)，(&Array[3],5,4)。当参数为(&Array[0],8,4)时，子函数ModifiedInsertsort( )里的 if(compare::lt(Array[j],Array[j-delta]))”语句比较的是 Array[4]与Array[0],此次比较正确。当参数为(&Array[1],7,4)时，子函数比较的仍然是 Array[4]与Array[0]，而此时应当比较 Array[5]与 Array[1]。当参数为(&Array[2],6,4)和(&Array[3],5,4)时，比较的仍然是 Array[4]与 Array[0]，而正确的比较应当分别为 Array[6]与Array[2],Array[7]与 Array[3]。难道是我哪里理解错了？

答：教材的算法是正确的。当参数为(&Array[1],7,4)时，注意 ModifiedInsertsort 函数的形参 Record Array[]数组的首地址被置为&Array[1]，即 Array[1]的地址，所以比较的还是Array[5]与 Array[1]。

在 ShellSorter（）中传递的就是 Array 中第 j 位元素的地址。在这里 j 的的范围从0~delta，表示以 delta 为增量，各个子序列的首地址。在 ModifiedInsert 中 j 表示从 i 到当前子序列首地址之间的元素，所以在插入排序中对 Array[j]与 Array[j-delta]进行比较。j 仅仅是一个变量，在两个函数中表示不同的含义。

(3) 问. Shell 排序 ModifiedInsertSort( )函数不采用优化后的插入排序算法(算法 7.6)而是采用算法 7.5 每次比较后都要交换呢?

答：优化后的插入排序算法，只有在数据规模很大时才有优势。在 shell 排序中有很多小子串处理，因此优化插入没有明显优势。shell 的分区思想才最关键。有一些教材还用冒泡排序来对子串处理。

36.快速排序的分区结果唯一吗？

问：设有关键码序列(Q,H,C,Y,Q',A,M,S,R,D,F,X)若采用以第一个元素为分界元素的快速排序法，则一趟扫描的结果是(F,H,C,D,Q',A,M,Q,R,S,Y,X)还是(F,H,C,D,M,A,Q,S,R,Q',Y,X)?

答：快速排序的具体结果取决于所使用的分区算法。如果限定用教材第 223 页那样的算法，那答案就是(F,H,C,D,M,A,Q,S,R,Q',Y,X)。注意，教材对于那些等于轴值的那些元素，都要移动，而且可以在轴值左边或者右边出现。

如果修改教材第 223 页 Partition 算法，对于那些等于轴值的那些元素都不移动，那么答案就是(F,H,C,D,M,A,Q,S,R,Q',Y,X)。

如果限定用教材第 221 页倒数第二段介绍的方法，同时从两头夹逼，找到一对逆序元素，则交换它们。例如，首先发现的逆序对是(Y,F)，交换之；然后发现(Q’, D)，交换之；左右两个下标汇合、交叉到 A、M，则交换(Q, M)。最后，得到的答案则是(M,H,C,F,D,A,Q,S,R,Q’,Y)。

37.系统函数 qsort 调用的比较函数问题

问：qsort 调用中的比较函数为什么要用下面两个函数中的后者？原先我把排序的比较函数写成

```
int cmp( const void*elem1, const void*elem2 )
        {    return ( *((int*)elem1) < *((int*)elem2) ); }
```

这样它只会返回小于时 1，否则 0；不会有等于的情况。而实际上应该有 1，－1，0 三种比较结果。所以排序出错。

改成

```
int cmp( const void*elem1, const void*elem2 )
{     return ( *((int*)elem2) - *((int*)elem1) );  }
```

就不错了。

答：查阅 MSDN 可知，qsort 需要的比较函数_stricmp 应该有下面特征：

当 elem1<elem2 时，返回<0;

当 elem1=elem2 时，返回=0;

当 elem1>elem2 时，返回>0;

可以看到 return (*((int*)elem1) < *((int*)elem2))只有在当 elem1<elem2 时返回 1，其它情况返回 0；

而 return (*((int*)elem2) - *((int*)elem1))刚好满足条件。

38.教材第 231 页优化的两路归并排序

问：“if (right <= **left) return;”冗余吧？**

答：此句看起来似乎多余，因为 Sort 函数中 right-left+1 <= THRESHOLD 就会进行直接插入排序，不会出现 right <= left 的情况。这一行代码显得有点多余。不过 THRESHOLD 可能不一定取 16，客观上也可以防止阈值取比较怪的值如－1 等意外的发生。

编程序的时候，很多时候我们逻辑上判断是不会出现某种可能性，但我们还是不遗余力的加上一个判断，就是为了最大限度的防止可能出错。

比如当我们为一个指针 new 空间之后，int *p = new int。容错性比较好的代码，通常会加一个判断，判断空间是否分配成功，如果空间分配不成功就会转入一个异常处理，尽管我们知道对于近似无限大的虚拟内存而言，这种异常出现的可能性微乎其微。

问：优化后的算法是不用像算法 7.14 那样需要检查子序列是否结束. 但它增加了比较的次数. 算法 7.14 中当一个子序列的元素插入完后,另一个子序列中的元素直接插入到 Array[]中,而不需要比较.

例如 原序列为: 12 37 49 41 30 71 58 19 65 15

先分为 2 个子序列

(12 37 49 41 30) (71 58 19 65 15)

对 这两个子序列分别再分解,然后分别归并,

归并后的序列为:

(12 30 37 41 49) (71 65 58 19 15)(颠倒过来后)

在插入元素 49 后 Array[] 为:12 15 19 30 37 41 49, 此时 index1 移动到元素 71 下, index2 在元素 58 下,然后还得分别比较 71>58 和 71>65 和 71>=71 (比较三次)来插入元素 58 65 71, 在算法 7.14 中,此时左边子序列已空了,右边子序列的三个元素无需比较直接插入 Array[]中就行了

而且在数组大于 16 时才进行这种归并,当数据量很大时,这种额外的比较开销对算法的时间性能没有影响吗?

答：问题提出得很好，但你的理解和思考还不够深入。教材的优化算法 7.15 确实比 7.14 快。比较算法 7.14 和 7.15，首先，二者的赋值运算都是一样的，对于长度为 n 的待排序序列，merge 一次则赋值 2n 次（一轮倒入临时数组为 n 次赋值，一轮每个值都从临时数组归并到位为 n 次赋值）。不同的是比较次数；

（1）在处理完一个子串之前

算法 7.14 每次合并操作都有三个比较运算：

while ((index1 <= middle) && (index2 <= right))

和 if (Compare::le(TempArray[index1], TempArray[index2]))

而 7.15 在处理完一个子串之前，每次合并操作都只有一个比较运算

if (Compare::le(TempArray[index1], TempArray[index2]))

这里的运算代价显然 7.14 高于 7.15

（2）某一个子串处理完之后

算法 7.14 对每次合并操作都有一个比较运算，“while (index1 <= middle)”或“while (index2 <= right)”（二者取一）

算法 7.15 对每次合并操作也有一个比较运算，if (Compare::le(TempArray[index1], TempArray[index2]))

这里的运算代价 7.14 与 7.15 相同

结论：Sedgwick 算法确实是优化了。

39.感觉书上的桶式排序 p234 的基数取法有点问题

答：首先肯定你敢于质疑的精神。但是，区间为 [0，max). 例如 max = 10，则数据为 0,1,2,..., 7,8, 9。结论，教材 P234 正确。

40.P243 的静态链表的基数排序第一趟分配不一致

问：第一趟分配结果

1-41-31

8-58-88

答：这里确实有插入顺序不一致的问题。当时举这个例子，想说明“LSD 的基数排序第一趟可以不稳定”，LSD 第二趟以后的排序必须稳定。当然，如果作为教材第 242 页算法的示例，应该统一。建议修改为：

8-88-58

问：在 Sort()函数中的第一个 for 循环中,i 从 0 到 i<n 为什么先用 for 循环使 Array[n-1]=i+1; 再在下面加语句 Array[n-1]=-1; 为什么不把 for 循环中的循环条件改为 i<n-1 呢？这是算法错误吗?

答：教材算法没有错误，你那样修改也是可以。二者效果是等价的，只是赋初值的习惯不同而已，教材算法多赋一次初值。

问：在 P241 中的 Collect( )函数中,函数首部括号中的参数 int i,应该为 int j 吧,不然与下面的注释 j 没有用到矛盾.

答：至于那个 j，则是早期版本中的一个冗余。抱歉没有及时修改注释，注释改为 i 就可以。总之，不影响算法功能。

问：在此函数中的第二个 while 循环中,循环条件为 k<r-1,我认为应该是 k<r 才对,要对 r 个队列进行收集啊,要不 P243 的第一趟分配图中,队列 queue[9]中的刻录 59 不能补收集(r=10,k<r-1=9,只处理了 queue[8])。请问是教材错了,还是我理解错了?

答：这位读者理解错误。这里的 k 是提前取了。

在 P243 倒数 L9-6 行，先把第一个非空的队列取到。

而 while (k<r-1)一进去，就 k++了，因此保证能取到 queue[r-1]。在本课程网站上有完整的源代码，你可以跟踪程序的执行，以帮助理解。

41.外排序多路归并时，如果某个顺串处理完了怎么办？

问：在利用败者树进行多路归并时，如果初始顺串的个数多于归并路数，而且在归并时，其中一个顺串中的数据最先被全部送入输出文件中，这时归并路数将减一吗？或者这么说，另外在输入文件中的顺串，会补入这个归并路中吗？还是统一放到下一趟归并时处理？

答：如果一个顺串的数据全部输出，应该在该处填充“无穷大”，你可以理解为归并路数减一。不能由其他顺串来补位，因为其他顺串与刚处理完毕的顺串之间的相对顺序不明确。长度为 n1, n2, ..., nk 的 k 个顺串，在 k 路归并后，形成一个长度为(n1+ n2+ ...+nk)的更大顺串，请参考教材 290-291，图 8.17。

42.折叠法散列函数的理解

问：对序列 21000442205864:用分界折叠的方法是:5864+0224+0004+1200 呢？还是 5864+0224+0004+0012 呢？

答：选前一种。验证折叠法时，可以把这些数字写在比较透明纸上，然后把纸折起来看效果。

43.ELFhash 是什么意思，我看不懂。

答：ELFhash 运算使得散列的结果受到字符串中每个字符的影响。那是给机器看的，不需要记忆和理解，也不需要手工计算结果。

44.如果中英文混编，那么读入时该如何区分呢？

答：读入的时候如果字节的最高位为 0，就是字母，也包括下划线等等其他符号。如果是 1，就是汉字，以及其他特殊符号，需要连读两个字节。

45.关于 P320-P323 散列表的一系列操作

答：比较运算根据算法的需要，有些是 KEY-KEY 型，有些是 KEY-ELEM 型，有些是 ELEM-ELEM 型。

算散列地址，基本是通过 KEY 来算，如果传入参数为 ELEM，那么使用 getkey(ELEM)来获得 KEY 值，然后 h()计算地址。

46.对中文词建立散列函数，要转化为数字么？

可以将一个汉字的两个字节，高字节放在高位，低字节放在低位，拼成一个数。

```
#include <iostream.h>
int main(int argc, char* argv[])
{
    short int si,high,low;//2 byte
    char highchar,lowchar;//1 byte
    cout << "input a chinese character" <<endl;
    cin >> highchar;
    while(highchar!='q'){//enter 'q' to quit
        if(highchar&0x80){//is chinese?
            cin >> lowchar;
        high=highchar;
        high<<=8;
        low = lowchar;
        low = low & 0x00ff;
        si=high|low; // get the key of character
        cout << si <<endl;
    }
    cout << "input a chinese character" <<endl;
    cin >> highchar;
    return 0;
}
```

47.排序和检索题意

(1) 习题 7.2 的解答

问“学习指导的对习题 7.2 的解答解法一中，图解中对直接插入排序算法的交换次数没错,但二分插入排序(算法 7.7)没有交换啊,只有 TempRecord = Array[];和 Arrat[] = TempRecord;两次赋值操作,而且是在每次外循环时都各执行一次,当然最后的 for 循环里移动的次数每次可能不同.但在解答中确也分比较和交换来分析就不对了吧,我感到在二分插入法中没有交换,只有移动,直接插入是交换(swap), 一次交换为三次赋值(移动).应该分为比较和移动来分析较合理. 解法二是正确的,而且与解法一是矛盾的.

答. 习题集给出两种解答的意思就是这两种都没有扣分。有些教材的二分插入，找到位置后，也用不断两两交换直到插入合适位置。

(2) 关于习题 7.20

问：$\log_3 n$ 和 $\log n$ 时间代价相比只差一个系数 $\log_2 3$？那么他们的时间代价是否一样？

答：对，它们的时间代价是一样的。

(3) 习题 7.28 如何推理

答：请查看 DONALD E.knuth "The Art of Computer Programming" 《计算机程序设计艺术》第 3 卷。

(4) 教材第 293 页习题 8.13

答：题目中要加限制，败方树的外部结点数 K=8，后面的"..."不要，只需要操作题目中给出的那些数据。

假定内存中可存放 K 个记录及在此基础上所构成的败方树，并且输入／输出操作是通过输入／输出缓冲区进行的。

利用败方树生成初始顺串方法的基本思路如下：从输入文件取 K 个记录，并且在此基础上建立败方树，将全局优胜者送入当前的初始顺串，并从输入文件取下一个记录进入败方树以替代刚输出的记录的结点位置。

若新进入败方树的记录的关键码值小于已输出记录的关键码值，那么，该新进入的记录不属于当前初始顺串，而属于下一个初始顺串，它不再参加比赛，这样就不会送到当前的初始顺串中；其他(属于当前初始顺串)的记录继续进行比赛。

若新进入败方树的记录的关键码值大于或等于已输出记录的关键码值，那么，该新进入的记录属于当前的初始顺串，它与别的属于当前初始顺串的记录继续进行比赛。

比赛不断进行，直至败方树中的 K 个记录都已不属于当前初始顺串，于是当前初始顺串生成结束，开始生成下一个初始顺串(即把下一个初始顺串称为当前初始顺串)。败方树中的 K 个记录重新开始比赛，这 K 个记录都属于新的当前初始顺串，因而都参加比赛。这样，一个初始顺串、一个初始顺串地生成，直到输入文件的所有记录取完为止。

(5) 感觉习题集第 237 页，习题 8.13 有问题

问：第一次建树，因为 7 最小，L[3]为空，而 10〉9，所以 10 在 L[3]的位置；

第二次建树，因为 10 最小，所以输出 10，L[3]为空，9 小于 10，所以 9 先不进行处理，放在缓冲区；20〉10，放在 L[3]的位置。

经过以上几步的处理，和输出串的刚开始几个元素都队的上号，但是，处理 18 的时候，却和答案不太一样，因为按照上面的处理方法处理到 17 时，此时缓冲区元素为 36，22，24，18，26，90，100，因此，我觉得 18 应该输出到第一个顺串中。

答：习题集答案是正确的。是你的理解与教材的不同。实际上，9 小于 10（10 是刚输出的数据），那么应该在 L[3]放入一个“无穷大”标记，L[3]就此退出比赛，9 呆在[3]位置，等待 L1、2、4-8 与它一样都有“无穷大”标记，他们一起构建第 2 个顺串的基础败者树。请你试着用这个方法，就可以得到与习题集一致的结果了。

你的想法还是很有创意的。问题是，“9 先不进行处理，放在缓冲区”，这些放入缓冲区的待处理数据数目很不明确，似乎不如习题集的方案好。

(5) 上机题 9.2 的题意

可以不用中文切词，但是需要建立索引。在检索关键字的时候需要一定的技巧。比如检索“明天”，根据索引找到“明”出现的文档，然后看后面一个字是否是“天”。

这个题目比较灵活，最低要求是能在单个文档进行“与”，“或”，“差”等操作，想做的比较好需要对一个目录下的所有文档进行上面的操作。

(6) 上机题 9.3 的题意

本题是桶式散列的一种改进方案。桶式散列有意让那些记录都散列到一个桶中，给选择散列函数增加了难度。本题的改进在于它可以充分利用散列函数的均匀分布特性。

本题两种方案：

1. 散列函数用 key(R)%197。不过，最后 19 个槽空着不能作为基地址，当然其中第 197、198、199 这三个槽还是可以被探测到。最后两个桶中第 200 - 215 这 16 个槽作为溢出区。溢出区满则报错，退出(当然也可以设计更大的散列空间，重新构造散列表) 。

2. 散列函数用 key(R)%213 (这意味着表中最后 3 个槽不作为任何记录的基位置) 。

216 槽开始作为一个不限制长度的溢出区（题目中没有限制溢出区的大小）。

冲突解决方案是在桶内回转的线性探查。例如，如果一个记录散列到第 5 个槽，第 5 个槽属于第 0 桶。散列函数会在第 0 桶内以 5，6，7，0，1，2，3，4 的线性顺序来探测，以解决冲突。如果第 0 桶满了，就把这个记录放到文件溢出区域。

同一个桶溢出的记录需要互相链接起来形成“溢出同义词表”，而且主桶中要给出溢出区同义词表表头的指针。

溢出区拉链有些类似于 ISAM（第 338 页图 10.5）。

六、索引和高级树结构问题

48.一级索引指向的是一个记录还是一个扇区？

答：稀疏索引指向每一个扇区，稠密索引指向每一个记录或者对象。计算时如果题目没有特别说明，可以理解指向为一个扇区吧，这样利用一级索引数可以得到扇区数，根据每个扇区的记录个数可得到总的记录个数，正如习题 10.1 , 10.2 那样。

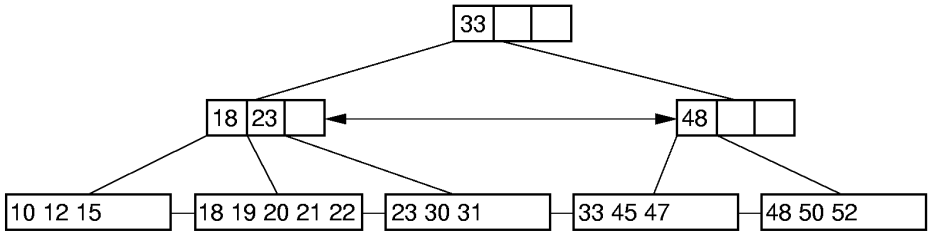
49.B 树/ B^+树为什么那样定义？

答：B 树/ B^+树是平衡结构。规定所有的叶结点在同一层可以，避免出现失衡的情况，从而使得对叶结点的访问时间一致，从而整体性能较好。

每个结点占用一个磁盘块。除根结点和叶结点外，B 树/ B^+树其他的结点有$\lceil m/2 \rceil$至 m 个子结点。允许结点不完全充满，使得插入/删除操作比较灵活。规定至少有$\lceil m/2 \rceil$至个子结点，意味着这些结点至少充满一半的磁盘块空间，这样可以保证树的层数较矮。因为对树形结构的索引，访问时间跟树的层数成正比，如果树越矮，那么访问时间就越短。B 树/ B^+树非根的内部结点有$\lceil m/2 \rceil$至 m 个子结点，可以折衷存储效率和操作效率。

如果根结点不空，也不是独根，B 树/ B^+树根结点的子结点数据范围在 2~m 之间。这是因为根结点分裂时，很可能分为 2 个子结点（根结点不可能只有一个子结点，这样太浪费），如果要求与其他内部结点具有同样的出度，需要对 B 树进行比较大的调整，这就违背了 B 树操作的局部调整原则。而且，实际运行时，根结点往往在内存中保存（当然最终它还是要写回外存，因此也有 m 的上限），不需要遵守内部结点的子结点数目下限 2 的约束。

实际上，B^+树还有其他的定义方法。例如，像张铭、刘晓丹翻译版《数据结构与算法分析》中定义的那样，其内部结点组织为 B 树，也就是有 k 个子结点的内部结点正好有 k-1 个关键码；其内部结点和叶结点允许的关键码个数限制范围叶可以不一样。这种 B^+树图例如下：



50.教材 p347 B 树关键码个数问题

问：教材 p347 图下面第 5 行：“关键码个数小于⌈m/2-1⌉”，我认为应该是“小于⌈m/2⌉”。

答：教材是正确的。在 m 阶 B 树中，子结点个数在⌈m/2⌉至 m 之间，关键码个数在⌈m/2-1⌉至 m-1 之间。教材讨论的是删除时下溢出的情况，本来结点中只有⌈m/2-1⌉个关键码，删除一个指定关键码，导致关键码个数小于⌈m/2-1⌉，需要借关键码或者合并结点。

51.关于极限情况下 B+树的删除

```
              15    35
       15     17          35
   10  15  16  17          35
```

在上图 2 阶 B+树中删除 35。会得到怎样的 B+树？

答：这个例子很好，因为它的下溢出出现了空结点的情况。最后删除的结果如何，关键在于如何定义对于空结点的操作。

情况 1：如果定义删除一个关键码后，结点为空，则删除父结点中对应关键码，关键码个数减一。于是在这道题中，删除叶结点中的 35 后，成为空结点，于是父结点也删除 35，也成为空结点，则根结点继续删除 35，则只剩下关键码 17。判断出根结点中只有一个关键码，于是最后删除根结点。则最终结果为：

```
              15  17
        10 15       16 17
```

情况 2：如果不对空结点作特殊规定，则删除叶结点中关键码 35 后，导致下溢出，无法从亲兄弟借结点或与亲兄弟合并（亲兄弟不存在），它的父结点的的关键码个数减为 0，删除 35，又导致它下溢出。但这时它可以向左兄弟（15, 17）借过来 17，根结点中的索引 35 可以修改为 17 也可以保持不变。最终调整为：

```
              15*  35
          15        17
     10   15        16   17
```

结论：如果按照尽量减少索引深度的原则来说，按照情况 1 的原则比较好。在实际应用中，B+树的阶都比较大，不会出现这种情况。

52.B 树和 B^+树的读写次数？效率怎么计算？

答：具体情况具体分析，看题目怎么出。

如果是 B 树，可能涉及到一个平均的问题， 因为关键码可能分布在非叶结点上，所以看具体题目的要求来处理；教材第 355 页给出了一个 B 树性能分析的例子，类似地进行分析就可以了。

如果是 B＋树，那么所有记录的磁盘地址指针都放在叶子结点上， 那么要读一个记录，并且所有索引结点都不在内存的时候，自然是算出 B^+树的高度，顺着树根找到叶子，从叶子中找到记录在磁盘的位置，然后从磁盘中直接读出；这样读盘次数是“树的层数”（树根为第 1 层）。

如果考虑读记录的那一次操作，则 B^+树读盘次数是“树的层数 ＋1”（1 表示最后读记录数)。如果考虑读记录的那一次操作，B 树读操作也要加 1。

53.教材上关于叶结点的表述

问：教材上在 B 树部分，关于叶结点的表述好像有点乱。首先在教材第 347 页图 10.9 下面有一句话说“如果删除的关键码不再叶结点层”；然后，又在 355 页第七行说“设 B 树包含 N 个关键码，因此有 N+1 个树叶”，我们该怎么理解？

答：早期很多教材认为 B 树的扩充外部结点是叶层。因此，教材中叶结点的表述确实有点不一致。应该把在教材第 355 页第七行修改为：“设 B 树包含 N 个关键码，因此有 N+1 个**扩展的外部结点，这些外部结点**都在第 I 层。”

54.教材上关于 B/ B^+树层数的表述

答：不同的教材对于树的层数定义不太一致。

大多数教材中，树（二叉树、树、B 树等）的第 0 层往往就是根结点。

例如在教材第 355 页第九行，“第二层至少有 $2\left\lceil \frac{m}{2} \right\rceil$ 个结点”的意思是：

第 0 层： 1 个结点，为根；

第 1 层： 至少 2 个结点；因为按照 B 树的定义，根至少有两个结点

第 2 层： 至少 $2\left\lceil \frac{m}{2} \right\rceil$ 个。

55.关于 347 页图 10.9 的疑问

问：插入关键码 460，向上调整时，第二层相应结点应该是 560（而不是 490）插入到根结点中吧。

答：插入了 460 之后，那么叶结点变成 393,396,400,435,460,471,将中间的 400 或者 435 放到父结点。

假设我们选择小的放到父结点，这里是 400，那么父结点变成 392,400,490,560,631,670，又满了，分解。还是从中间两个中选择小的放到父结点，这里是 490。

应该说选择中间两个数任何一个都行，不过必须一致，一直选择小的，或者一直选择大的。

56.关于习题集 292 页习题 10.12 的疑问

问：第十章课后习题 10.12 ，内部结点最多存 50 个关键码，那一个结点只有在有 51 个关键码时才要分裂,这时分裂出的两个子结点应该共有 51 个关键码，为什么书上按 50 个计算呢？

答：习题集答案是正确的。你在计算时，只考虑了由插入而分裂产生结点的情况。实际上，还应该考虑删除记录所造成的影响。具体分析请参考习题集 10.11 解答中“典型错误”分析。

57.第 463 页半伸展树的疑问

问：教材第 463 页给出的那个半伸展树的例子总觉得不对。图 12.47（b）共有 4 个图，按照书上给出的算法，开始时 T 指针指向“T”结点，则“S”绕“R”旋转以后变成了第二个图，此次操作正确，在执行了“T=T->parent”后，T 指针指向“S”结点，然后是“q”绕“p”旋转，变成了第三个图，此次操作也正确，在执行了“T=T->parent”后,T 指针指向“q”结点，此时，按照算法，"q"已经是根结点了，应该结束，然而书上却又旋转了一次，变成了第四个图，这是为什么呢？

答：图 12.47(b)的文字标注错了（勘误表中已改正），本来是连续两次访问 T 的情况。前三个子图为第一次访问 T 时发生两次旋转的示意。第四个子图表示对子图三第二次访问 T 后的情况。

七、C++常见问题

58.using namespace std 是什么意思？

这条语句被称为 using 指示符，其中 using 和 namespace 都是关键字。标准 C++库中所有组件都是在一个称为 std 的名字空间中声名和定义的。这条语句告诉编译器下面要使用在名字空间 std 中声名的名字。

为了增强程序的可移植性，建议大家使用标准 C++的库，即 STL。

59.关于 C++的头文件

如果想调用库中的文件，就使用尖括号(<>)，如果想调用自己定义的头文件，就使用引

号(“”)。标准 C++的头文件没有后缀，带有后缀如.h 的头文件不是标准 C++的头文件。如果要用 STL 中的类如 string 等，应该如下引入：

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

如果要使用旧库中的字符串类，应该如下引入：

```
#include <string.h>;
```

如果要使用自己定义的字符串类，应该如下引入：

```
#include “string.h”;
```

下面是一些常见的问题。

(1) <stdlib.h>是干什么的？

答：<stdlib.h>和<string.h>类似，是 C 语言的一个 library,包含一些常量的声明和常用的函数，比如 rand()函数（产生随机数）等。

(2) #include <string.h>后，声明字符串变量 “string s1”会报错 undeclared identifier？但是其中的一些函数如 strlen 却可以使用？难道要自己编写 string 类？

答：(a) 如果没有编写自己的 string 类，#include <string.h>之后，不能直接声明 string s1。因为<string.h>是系统的 library，里面并没有定义 string 类，只是定义了一些 char*字符串的常用函数（其中也包括 strlen()），书上有自定义的 string 类，可以自己在上机试验，注意需要#include "string.h"（注意，是“”而不是<>）。

(b)事实上，可以直接引用 STL 中定义的标准 string 类。首先要“#include <string>”，注意，没有“.h”。然后，需要

```
using namespace std;
```

或者声明 using std::string;

这样，定义 string s1 就不会出错了。不过请注意，教材第 62 页自己定义的 String 类与 STL 的 string 类略有差别。

(c) 另外，教材上定义的 strlen()是 String 类的 Public 函数，必须针对一个 String 对象引用，单独引用 strlen()则是调用标准 C 的函数。

```
String P;  int len
len = strlen(P);  或者  len = P.strlen();
```

60.如何通过 C/C++命令运行一个文件？

问：比如我编了一个可执行程序，存放在某个目录下。要在程序中调用该程序以执行某些操作，应该如何进行？

答：有两种方法

（1）用 system 命令

```
#include<windows.h>
system("notepad.exe a.txt");
```

就可以打开 a.txt 文件了

不过要是使用文件绝对路径时要记得改成双斜杠

比如 F:\a\a.txt 要写成 F:\\a\\a.txt

（2）还有一个 ShellExecute

可以使用 Windows API 函数 WinExec、ShellExecute。这两个函数可以调用 Windows 和 DOS 程序。WinExec 主要运行 EXE 文件。例如：

WinExec("Notepad.exe Readme.txt", SW_SHOW);

ShellExecute 不仅可以运行 EXE 文件，也可以运行已经关联的文件。例如：

ShellExecute(0, "open", "http://askpro.yeah.net", NULL, NULL, 0)

61.怎样使用 STL 中的队列啊？

问：为什么按照课本上的 using std::queue; 总是报错说 'std' : is not a class or namespace name 呢？

答：需要先#include <queue>。

62.关于 STL 中的 top

问：教材第 94－95 页二叉树非递归前序和后序周游的算法中 为什么用到 aStack.top();操作例如：我觉得 pointer=aStack.pop(); 就完全可以代替 pointer=aStack.top(); 和 aStack.pop(); 这种写法是问什么呢？是为了使算法更容易看懂和理解？

答：这两个算法中都用到了标准模板库 STL 中关于堆栈的函数。其中 top 函数表示取栈顶元素，并将结果返回给用户；pop 函数表示将栈顶元素弹出（如果栈不空的话），pop 函数仅仅是一个操作，并不将结果返回。因此，表达式 pointer=aStack.pop()在 STL 中是不合乎语法的。

事实上，有些库函数提供了这样的函数 ptop，它的意思是首先取栈顶元素返回给用户，然后将栈顶元素弹出，这个函数事实上就实现了你的功能。STL 之所以将这两个操作分开，主要是概念上显得清晰，保证一个函数只确定地完成一项特定的功能，使得函数之间的耦合度降低。

63.请问如何获得某段代码执行耗时？

答：代码执行之前记录时间，结束时记录时间。两个时间相减就可以。

记录时间的函数很多，不同库里有不同，可以查手册。除了课程网站上源代码（讲义上也给出了）那样的计时方法外，还有以下几种方法。

1）C++里可以用_ftime()_ftime()可以精确到毫秒

2）MFC 程序通常使用下面的方法得到当前的日期和时间：

```
CTime t = CTime::GetCurrentTime();
```

CTime 类使用起来是非常方便的。

3）VC++也支持标准的 C 语言中使用 time 函数获得当前的日期和时间，在帮助中有完整的例子。使用 API 函数 GetLocalTime 也可以获得当前的日期和时间。如：

```
SYSTEMTIME    t;
GetLocalTime(&t);
```

可以像下面这个程序段这样来获取时间：

```
#include <windows.h>
int t = GetTickCount();
// 此处调用需要计时的用户代码
```

```
cout << GetTickCount() - t << endl;        // 以毫秒为单位
```

64.文件流操作

C++的输入/输出功能由输入/输出流(iostream)库提供，要想使用 iostream 库中的程序必须包含相关的系统头文件：

```
#include <iostream>;
```

如果涉及到文件的输入输出，除了上面头文件之外，还有引入：

```
#include <fstream>;
```

当然，在引入这两个头文件之后，不要忘记写上：

```
using namespace std;
```

(1)文件的打开与关闭

为了打开一个输出文件，必须声明一个 ofstream 类型的对象，参数是文件名：

```
ofstream outfile("name_of_file");
```

为了测试是否成功打开了这个文件，可以写：

```
if (!outfile)
    cerr << "cannot open this file. "<<endl;
```

类似地，为了打开一个文件供输入，必须声名一个 ifstream 类型的对象：

```
ifstream infile ("name_of_file");
if (!infile)
    cerr <<"cannot open this file"<<endl;
```

下面是一个简单的程序，从键盘获得输入，并输出到 record.txt 中，然后再以 record.txt 作为输入，并输出到屏幕。断开与文件的连接，用成员函数 close()。

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ofstream outFile("record.txt", ios_base::app);//打开文件 record.txt
    if (!outFile) {          //检验是否被成功打开
        cerr <<"cannot open this file."<<endl;
        return -1;
    }
    char ch;
    while (cin.get(ch))   //从键盘获得输入
        outFile.put(ch); //输出到 record.txt

    outFile.close();
    ifstream inFile("record.txt");    //将此文件作为输入
    if (!inFile) {          //检验是否被成功打开
        cerr <<"cannot open this file."<<endl;
        return -1;
    }
```

```
    while (inFile.get(ch))   //从 record.txt 中获得输入
        cout.put(ch);   //输出到屏幕
    inFile.close();
}
```

(2)文件的打开模式

可以在上面的程序中注意到，ofstream 的两个参数分别指定了要打开的文件名和打开模式。程序中使用的打开模式是附加模式(ios_base::app)，与此模式相对的模式是输出模式(ios_base::out)。输出模式将原来文件中所有数据都丢弃，而附加模式把新数据添加到原来文件的尾部。

fstream 对象常见的打开模式列出如下：

ios_base::app： 以追加的方式打开文件

ios_base::ate： 文件打开后定位到文件尾，ios_base:app 就包含有此属性

ios_base::binary： 以二进制方式打开文件，缺省的方式是文本方式。

ios_base::in： 文件以输入方式打开

ios_base::out： 文件以输出方式打开

ios_base::nocreate： 不建立文件，所以文件不存在时打开失败

ios_base::noreplace：不覆盖文件，所以打开文件时如果文件存在失败

ios_base::trunc： 如果文件存在，把文件长度设为 0

(3)文件的读写

在上面的例子中，用到了 fstream 的成员函数 get(char)和 put(char)用于文件的输入和输出。更为常用读写函数是：

read(char_type *_*Str*, streamsize _*Count*)

write(const char_type *_*Str*, streamsize _*Count*);

其中读函数把从文件读来的字符放入_Str 中，写函数把_Str 中的内容写入文件。

对文件重定位可以用成员函数 seekg()或 seekp()，其中 seekg()用于在输入流中重定位，seekp()用于在输出流中重定位。以 seekg()为例，例如

```
fstream io ("word.out", ios_base::in);
io.seekg(6);
```

表示把 io 定位到 word.out 中的第 6 个字符。而

```
io.seekg(20, ios_base::cur);
```

表示把 io 定位到文件当前位置后面的第 20 个字符。

可见，seekg()的第一个参数用来指定偏移量，第二个参数用来指定文件是从当前位置还是开始位置还是文件尾部开始计算。第二个参数可以是下面三个值之一：

ios_base::beg 文件的开始；

ios_base::cur 文件的当前位置；

ios_base::end 文件的结尾。

STL 提供了丰富文件操作函数和操作方式，在需要时可以参阅 STL 的手册。

(4)下面是有关文件操作常见的问题。

问：在按照讲义用 ifstream ofstream 生成对象时，出现了如下错误：

```
error C2872: 'ifstream' : ambiguous symbol
```

答：这是一个二义性的错误：The compiler could not determine which symbol you are referring to，有可能是因为即包含了 fstream.h 又有语句 using namespace std 这样编译器无法判断你到底用的是 fstream.h 中的定义，还是名字空间的定义，于是产生二义性错误，

正确的定义应该如下：

＃include <fstream>

using namespace std;

或者 #include <fstream.h>

问：我用了#include<fstream> using namespace std; 来包含头文件。结果在fin.open(index,ios::in|ios::nocreate);这条语句时出错了。错误报告如下：

'nocreate' : is not a member of 'basic_ios<char,struct std::char_traits<char> >'

'nocreate' : undeclared identifier

'nocreate' : is not a member of 'basic_ios<char,struct std::char_traits<char> >'

我发现用#include<fstream.h>就没事了 这究竟是怎么回事啊？

答：#include<fstream> using namespace std;采用的新库中的 iostream，

#include<fstream.h>采用的是旧库中 iostream。

以下作为旧 iostream 库元素的函数、常数和枚举数不是新 iostream 库的元素：

filebuf、fstream ifstream 和 ofstream 的 attach 成员函数

filebuf、fstream ifstream 和 ofstream 的 fd 成员函数

filebuf::openprot

filebuf::setmode

ios::bitalloc

ios::nocreate

ios::noreplace

ios::sync_with_stdio

streambuf::out_waiting

streambuf::setbuf（相同的行为使用 rdbuf -> pubsetbuf）

其中包含你使用的 nocreate，因此会出错。

65.关于文本文件处理的问题

(1) 网页中可能有不规范的书写格式，如：

<a href

=

gh

>

x

g

它产生的效果和x g 是一样的,都是一个叫 x g 的连接(注意中间的空格)如果需要兼容类似这种不规范格式的话会带来很大的困难,请问是否可以把类似情况屏蔽掉,只处理标准格式.

答：链接分析时，可以将网页的内容都读入一个字串，然后对这个字串进行分析。对于不规则的链接，在程序中自行判断，并规范化即可。链接查找第一步可以查找标识，即<>中内容，然后判断是 href 的标识，之后再取 href 内容即可。

(2) 都读入一个字符串？这个串的长度**岂不是要到 100000??对于某些中文比较多的网页来说，根本不现实啊！**

答：链接都是"<a"开头的，可以先找到"<a",然后再找到紧跟其后的第一个">",将"<a" 到">"

的这一段字符串都截取下来，然后对这一个串进行分析，这些用 c++里的字符串处理函数都是可以实现的。

100000 长度不值得奇怪，最多 100k 而已。c++中的 string、MFC 中 CString、API 中的 String，都支持动态分配字符存储空间的。搜索引擎里的爬虫技术复杂的多，分析链接只是其中的一部分功能。练习这个作业的初衷，是应该让大家用数据结构中所学的知识来处理这个问题，爬虫算法思想是对的即可。

真正实用的 spider 或者 robot，处理起来，不必每次都将整个页面读入分析，按段分析也可以。这个是搜索引擎处理的细节和优化问题，不同引擎用的技术可能不同。真正技术内幕还真不好获取，大家可以上网去搜索相关的资料。

对于某些格式太不规范的链接，爬虫的态度一般是置之不理，真正实用的爬虫，不会为分析这些怪异链接地址而耗费时间的。至于 href 后的属性值是双引号、单引号、或者无引号方式，可以在程序中进行判断。

另外，很少网页制作者故意将代码写成这样的。如果作者想让自己网站的某些链接不被搜索引擎访问，大可不必这么写，按照 robot exclusion protocol 写一个 robot.txt 就可以了。

(3) eatwhite()函数为什么不能用啊？编译报错：

error C2039: “eatwhite” : 不是“std::basic_ifstream<_Elem,_Traits>”的成员。

我文件开头是

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

为什么会这样呢？

答：可能包含的库文件不对吧。试试#include<fstream.h>，不要用 using namespace std。

另外，其实一个一个字符分析就可以了，不会很麻烦的。

大体框架的伪代码如下：

```
while (!eof()) {
 while(getch() != '<');   // Find a Tag;
 while(getch() != ' ')
       Read_Tag_Name_Char_by_Char(strTag);
 if (strTag != 'a')          // Check if it is a hyperlink
       continue;
 while(getch() != '>') {   // Read all tag properties
    Ignore_Space_and_CrLf();
    while(getch() != '=')
       Read_Property_Name_Char_by_Char(strProp);
    if (strProp != 'href')            // Check if it is the URL
       continue;
    while(getch() != ' ' or '>')   // Retrive URL from property value,
                                   // be careful of quotation marks!
       Read_Property_Value_Char_by_Char(strValue);
    URLs[count++] = strValue   // Store the URL
 }
}
```

这只是框架，仅仅这么写会出很多问题的，比如没有值的属性、用引号括起来的属

性值中的空格等等，这些都要注意。用循环读任何东西的时候记住要不断忽略空格和回车，并判断文件是否结束。

66.关于网络爬虫中的文件路径处理

(1) 修改的是相对路径，将相对路径补全为绝对路径的时候，例如：绝对路径为 E:\project\1.htm，测试是提示\p,\1 为一个字符，也就是说\本身不是一个字符，请问为什么？ 怎么得到绝对路径？

答："\\" 是字符"\"的转义符，就像"\n"是回车的转义符一样。因为字符"\"被用作转移符的前缀了，为了输入"\"我们就必须也把它用一个转义符替代，路径中的\请用转义字符\\代替这样就可以的到字符\。

(2) 路径是不是还有左斜和右斜（就是'/' 和'\'），这有什么区别么？

答：在操作系统的目录文件的路径中，Windows 中，对"/"和"\"都支持，甚至可以混合用。编程中，反斜杠要用转义字符"\\"表示，正斜杠直接用"/"就可以。在 Uinx、Linux 下，一般只能用"/"。

八、考研问题

67.关于考研技巧问题，请到 BBS 考研版

请到 http://bbs.pku.edu.cn/cgi-bin/bbstop?board=Kaoyan 去询问。请不要给张铭发 email 询问考试重点、范围等问题。张铭要说的话，都体现在"2007 级硕士生入学考试数据结构复习大纲"http://db.pku.edu.cn/mzhang/ds/DSgradreview.HTM 中。

68.北大计算机系招生名额

问：每年北大招收计算机应用或者计算机体系结构的统考生名额最后剩下多少了，似乎十几个吗？那么每年又有多少人报考这几个专业呢？

答：北大录取的统考生比例每年都有所小幅度变动，每年报考计算机专业的人数也不同。

2005 年计算机专业

"统考"人数 625，最高分 413

70 名，355；80 名，351

84 名，349；96 名，346

105 名，343；120 名，340

去掉政治英语 55-，数学专业课 90-之后

人数 157，最高分 413

70 名 355；80 名，351

84 名，349；96 名，345

105 名，342；120 名，337

06 招生计划：
计算机系统结构 48（其中深圳 35）
计算机软件与理论 57（深圳 0）
计算机应用技术 41 （其中智能科学系 19）

05 录取情况：最低录取分数：345

	录取人数	统考	强军	免试	自筹	深圳
计算机系统结构	45	6		9		30
计算机软件与理论	72	28	1	34	2	7
计算机应用技术	57	19	6	31	1	

69.北大数据库方向招生名额

我们数据库方向在“081202 计算机软件与理论”下统一招生，软件与理论 2007 年共招 57 人。

数据库方向目前有 6 位硕士导师，具体名额还不能确定。可以肯定的是，每年我们都留出一些名额给统考学生。而且，据我所知，每年到数据库方向复试的学生，基本上留在计算机系或信息学院其他专业攻读硕士学位，我们还有北大深圳研究院硕士名额。我本人每年招收 1 名左右统考硕士生。

以下 5 个方向是本实验室在招生目录上设置的方向，第一志愿的保研学生以及考研学生的简历将送到本实验室筛选。每年 7-8 月，保研学生可以跟实验室老师联系。

20. 典型应用领域的数据库与信息系统
21. W e b 环境下的数据库与信息集成
22. 联机分析处理与数据挖掘的方法和应用
23. 数据库系统实现技术
24. 语义网与数据网络

70.操作系统、离散数学、高数的复习大纲？命题老师的个人主页？

计算机软件基础（数据结构、操作系统）的复习大纲已经提交给学院了。请关注信息科学学院网站 http://eecs.pku.edu.cn/。

2007 年，离散与高数的分值比例为 90：60，数据结构与操作系统的分值比例为 80:70。

其他情况无可奉告。

71.所有报考计算机软件基础的试卷题目(代号 896)是不是一样的啊？

数据结构命题老师每年只参与一份试卷命题，阅卷时也没有发现不同的试卷。

72.北大对跨专业报考有什么限制吗？复试会不会加试呢？

答：据我所知，北大没有对本科就读院校、本科就读专业存在限制或者歧视现象。有些研究室可能在复试面试时，加试本专业相关的一些基础问题。我见过不少研究生转到计算机专业的学生，他们非常努力，也很成功。

73.您喜欢带什么样的学生？

答：我所接触的很多同事都认为，看待一名研究生，首先看其学习和工作态度是否认真，其次看其基本功（数学、逻辑、编程能力、中英文）扎不扎实。

九、鸣谢

感谢赵海燕、王腾蛟老师和历届助教的大力协助，全体信息学院同学的认真钻研、积极探索，所有热心读者的帮助。

