备考2010年计算机专业研究生考试通用教材

文都教育

2010年
全国硕士研究生入学统考
计算机学科专业基础综合
辅导讲义

2010NIANQUANGUOSHUOSHIYANJIUSHENGRUXUETONGKAOJISUANJIXUEKEZHUANYEJICHUZONGHEFUDAOJIANGYI

策划◎文都考研命题研究中心　　主编◎崔巍　卫真　白龙飞等

考纲要求提纲挈领
复习要点层次分明
内容讲述重点突出
例题设置紧扣大纲

计算机学科专业基础综合第一书

原子能出版社

备考2010年计算机专业研究生考试通用教材

文都教育

2010

全国硕士研究生入学统考

计算机学科专业基础综合

辅导讲义

策划◎文都考研命题研究中心

主编◎崔巍 卫真 白龙飞等

原子能出版社

图书在版编目(CIP)数据

计算机学科专业基础综合辅导讲义/崔巍等编著. —北京:原子能出版社,2009.7
ISBN 978-7-5022-4666-2

Ⅰ.计… Ⅱ.崔… Ⅲ.电子计算机—研究生—入学考试—自学参考资料 Ⅳ.TP3

中国版本图书馆 CIP 数据核字(2009)第 120538 号

计算机学科专业基础综合辅导讲义

总 编 辑 杨树录
策　　划 文都考研命题研究中心
责任编辑 刘　岩
特约编辑 师　潭　宫　静
印　　刷 环球印刷(北京)有限公司
出版发行 原子能出版社(北京市海淀区阜成路 43 号　100048)
经　　销 全国新华书店
开　　本 787mm×1092mm　1/16
印　　张 29.5　　**字　　数** 450 千字
版　　次 2009 年 7 月第 1 版　2009 年 7 月第 1 次印刷
书　　号 ISBN 978-7-5022-4666-2　　**定　　价** 50.00 元

网址:http://www.aep.com.cn　　**E-mail:atomep123@126.com**
发行电话:010-68452845

前　言

全国硕士研究生入学统一考试计算机科学与技术学科初试科目于2009年进行了调整，计算机学科专业基础综合（包括数据结构、计算机组成原理、操作系统和计算机网络）成为全国统考科目。通过对2009年真题的研究，编者发现：计算机学科专业基础综合考查的重点是考生对专业基础知识、基本理论、基础方法的掌握水平及分析问题、解决问题的能力。所以考生在复习备考此科目时应将精力放在基本概念、基本原理与基本方法的融会贯通上，并力求熟练运用所学知识分析、判断和解决有关理论问题与实际问题。这也是编者编写本书的依据之一。

为了帮助考生更好地把握计算机学科专业基础综合的复习要点，编者对《全国硕士研究生入学统一考试计算机学科专业基础综合考试大纲》规定的考试内容和考试要求进行了深入分析，并结合多年来对这些课程的潜心研究编写此书，以帮助同学们迅速抓住考试重点、掌握难点。

全书分为四个部分：第一部分数据结构，第二部分计算机组成原理，第三部分操作系统，第四部分计算机网络。每章内容包括考纲要求、复习要点。在考纲要求中明确本章的主要知识点，阐述清晰；在复习要点中对相关课程考纲的各个知识点进行集中讲解和提炼，以帮助考生有针对性的复习，并选择了典型例题进行分析，方便考生对每部分知识的考核方式有所把握，加强考生的应试能力。

本书具有以下特点：

1. 考纲要求提纲挈领。每一章以“考纲要求”开始，以便考生了解该章知识的考试要求，整体把握复习侧重点。

2. 复习要点层次分明。“复习要点”部分均逐层展开，脉络清楚，利于考生建立知识框架。

3. 内容讲述注重基础。知识点讲解以基础为中心，重视在基础中体现能力，充分体现大纲精神。

4. 例题设置紧扣大纲。为使考生充分掌握相关知识要点及考试出题规律而设的例题均围绕大纲要求编制。

参与本书编写的教师均为国家重点院校的长期从事计算机科学与技术学科相应本科生及研究生课程教学的一线教授和副教授，在相关课程中均具有十年以

上的教学经历，并先后编写过多本教材和教学参考书。本书数据结构部分由崔巍老师编写，计算机组成原理部分由蒋本珊老师编写，操作系统部分由孙卫真老师编写，计算机网络部分由白龙飞老师编写。全书由崔巍老师统稿。

在本书的编写过程中，参考了一些相关的书籍和资料，在此向这些书的作者表示深深的谢意。由于编者水平有限，时间也比较仓促，尽管经过反复校对与修改，书中难免还存在错漏和不妥之处，敬请广大读者和专家批评指正。

衷心地希望本书能帮助考生在考试中取得理想的成绩！

编　者

2009 年 7 月

目　　录

第一部分
数据结构

第一章 线性表

线性表是最简单、最基本、最常用的一种线性结构。它有两种存储方法：顺序存储和链式存储，它的主要基本操作是插入、删除和检索等。

★考纲要求

（一）线性表的定义和基本操作

线性表的逻辑结构，是指线性表的数据元素间存在着线性关系。主要是指：除第一个及最后一个元素外，每个结点都只有一个前趋和只有一个后继。

（二）线性表的实现

1. 顺序存储结构

（1）线性表的顺序存储结构，靠元素存储的先后位置反映数据元素的逻辑关系。

（2）在具体语言环境下有两种不同实现：表空间的静态分配和动态分配。

（3）用向量（一维数组）表示，即给定下标可以存取相应元素，属于随机存取的存储结构。

（4）线性表的顺序存储结构实现插入、删除、定位等运算的算法。

2. 链式存储结构

（1）线性表的链式存储结构，靠指针来反映数据元素的逻辑关系。

（2）链表的存取需要从头指针开始，顺链而行，不属于随机存取结构。

（3）几种常用链表的特点和相关算法设计：单链表、单循环链表、双向链表、双向循环链表的生成、检索、插入、删除、遍历、分解和归并等操作。

（4）从时间复杂度和空间复杂度的角度综合比较线性表在顺序和链式两种存储结构下的特点及其各自适用的场合。

3. 线性表的应用

运用顺序表和链表的特点解决复杂的应用问题。

★复习要点

1.1 线性表的逻辑结构

1. 线性结构特点

线性结构的特点是数据元素之间是一种线性关系：

（1）存在唯一的一个被称为“第一个”的数据元素；

（2）存在唯一的一个被称为“最后一个”的数据元素；

（3）除第一个之外，集合中的每个数据元素均只有一个前驱；

（4）除最后一个之外，集合中的每个数据元素均只有一个后继。

2. 线性表定义

线性表是一种线性结构，在一个线性表中数据元素的类型是相同的，或者说线性表是由同一类型的数据元素构成的线性结构，定义如下：

线性表是具有相同数据类型的 $n(n\geq 0)$ 个数据元素的有限序列，通常记为：

$(a_1,a_2,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n)$

其中 n 为表长，$n=0$ 时称为空表。

需要说明的是：a_i为序号为 i 的数据元素($i=1,2,\cdots,n$)，通常将它的数据类型抽象为 ElemType，ElemType 根据具体问题而定。

1.2 线性表的顺序存储结构

1. 顺序表

线性表的顺序存储是指在内存中用地址连续的一块存储空间顺序存放线性表的各元素，用这种存储形式存储的线性表称其为顺序表。因为内存中的地址空间是线性的，因此，顺序表用物理上的相邻实现数据元素之间的逻辑相邻关系是既简单又自然的。

设 a_1的存储地址为 $\mathrm{Loc}(a_1)$，每个数据元素占 d 个存储地址，则第 i 个数据元素的地址为：

$$\mathrm{Loc}(a_i)=\mathrm{Loc}(a_1)+(i-1)*d \qquad 1\leqslant i\leqslant n$$

这就是说只要知道顺序表首地址和每个数据元素所占地址单元的个数就可求出第 i 个数据元素的地址，这也是顺序表具有按数据元素的序号随机存取的特点。

在程序设计语言中，一维数组在内存中占用的存储空间就是一组连续的存储区域，因此，用一维数组来表示顺序表的数据存储区域，如下描述。

线性表的静态分配顺序存储结构：

```
#define LISTSIZE 100                    // 存储空间的最大分配量
typedef  struct{
    ElemType  elem[LISTSIZE];
    int       length;                   // 当前长度
}Sqlist;
```

在线性表的静态分配顺序存储结构中，线性表的最多数据元素个数为 LISTSIZE，元素数量不能随意增加，这是以数组方式描述线性表的缺点。为了实现线性表最大存储数据元素数量随意变化，可以使用一个动态分配的数组来取代上面的固定长度数组，如下描述。

线性表的动态分配顺序存储结构：

```
#define LIST_INIT_SIZE 100              // 存储空间的初始分配量
#define LISTINCREMENT 10                // 存储空间的分配增量
typedef  struct{
    ElemType  *elem;                    // 线性表的存储空间基址
    int  length;                        // 当前长度
    int  listsize;                      // 当前已分配的存储空间
}Sqlist;
```

2. 顺序表上基本运算的实现

(1)顺序表的初始化

顺序表的初始化即构造一个空表，这对表是一个加工型的运算，因此，将 L 设为引用参数。首先动态分配存储空间，然后，将 length 置为 0，表示表中没有数据元素。

【算法如下】

```
int Init_SqList (SqList &L){
    L.elem = (ElemType *)malloc(LIST_INIT_SIZE * sizeof(ElemType));
    if (!L.elem) exit (OVERFLOW);// 存储分配失败
    L.length =0;
    L.listsize = LIST_INIT_SIZE;        // 初始存储容量
    return OK;
}
```

算法 1.1

设调用函数为主函数，主函数对初始化函数的调用如下：

```
main(){
    SqList L;
    Init_SqList (L);
    ...
}
```

(2)插入运算

线性表的插入是指在表的第 i(取值范围:$1 \leqslant i \leqslant n+1$)个位置上插入一个值为 x 的新元素，插入后使原表长为 n 的表：

$$(a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n)$$

成为表长为 $n+1$ 的表：

$$(a_1, a_2, \ldots, a_{i-1}, x, a_i, a_{i+1}, \ldots, a_n)。$$

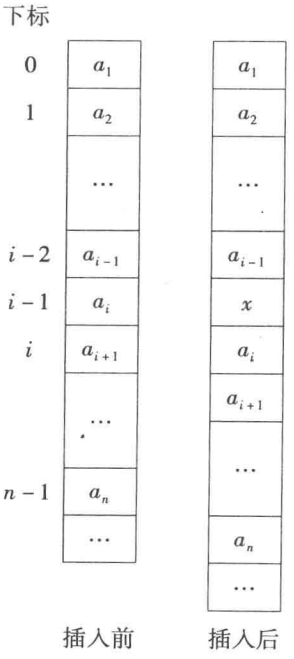

图 1-1-1　顺序表的插入

顺序表上完成这一运算通过以下步骤进行：

①将 $a_i \sim a_n$ 顺序向下移动，为新元素让出位置；(注意数据的移动方向：从下往上依次下移一个元素)

②将 x 置入空出的第 i 个位置；

③修改表长。

【算法如下】

```
int Insert_SqList (SqList &L, int i, ElemType x){
    if (i<1 || i>L.length+1) return ERROR;          // 插入位置不合法
    if (L.length >= L.listsize) return OVERFLOW;    // 当前存储空间已满,不能插入
        // 需注意的是,若是采用动态分配的顺序表,当存储空间已满时也可增加分配
    q = &(L.elem[i-1]);                             // q 指示插入位置
```

```
    for (p=&(L.elem[L.length-1]); p>=q; --p)
        *(p+1)=*p;                          // 插入位置及之后的元素右移
    *q=e;                                   // 插入e
    ++L.length;                             // 表长增1
    return OK;
}
```

算法 1.2

【性能分析】

顺序表上的插入运算,时间主要消耗在数据的移动上,在第 i 个位置上插入 x ,从 a_i 到 a_n 都要向下移动一个位置,共需要移动 $n-i+1$ 个元素。

由于 i 的取值范围为:$1 \leqslant i \leqslant n+1$,即有 $n+1$ 个位置可以插入。设在第 i 个位置上作插入的概率为 P_i,则平均移动数据元素的次数:

$$E_{in}=\sum_{i=1}^{n+1}p_i(n-i+1)$$

在等概率情况下,即 $p_i=1/(n+1)$,则:

$$E_{in}=\sum_{i=1}^{n+1}p_i(n-i+1)=\frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1)=\frac{n}{2}$$

这说明在顺序表上做插入操作需移动表中一半的数据元素,显然顺序表上的插入运算的时间复杂度为 $O(n)$。

(3)删除运算

线性表的删除运算是指将表中第 i(取值范围为:$1 \leqslant i \leqslant n$)个元素从线性表中去掉,删除后使原表长为 n 的表:

$$(a_1,a_2,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n)$$

成为表长为 $n-1$ 的表:

$$(a_1,a_2,\cdots,a_{i-1},a_{i+1},\cdots,a_n)。$$

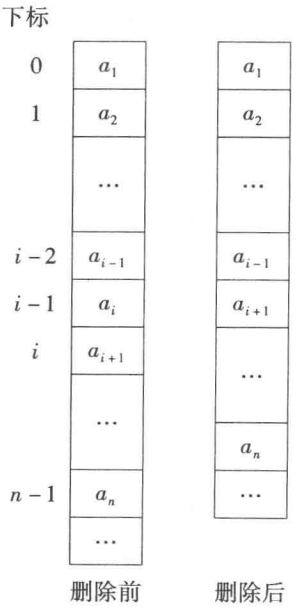

图 1-1-2 顺序表的删除

顺序表上完成这一运算的步骤如下：

①将 $a_{i+1}\sim a_n$ 顺序向上移动；(注意数据的移动方向：从上往下依次上移一个元素)

②修改表长。

【算法如下】

```
int Delete_SqList (SqList &L,int i) {
    if ((i < 1) ||(i > L. length))   return ERROR;  // 删除位置不合法
    p = &(L. elem[ i - 1 ]);                        // p 为被删除元素的位置
    e = * p;                                        // 被删除元素的值赋给 e
    q = L. elem + L. length - 1;                    // 表尾元素的位置
    for( + + p;p < = q; + + p)
      * (p - 1) = * p;                              // 被删除元素之后的元素左移
    - - L. length;                                  // 表长减 1
    return OK;
}
```

算法 1.3

【性能分析】

顺序表的删除运算与插入运算相同，其时间主要消耗在移动表中元素上，删除第 i 个元素时，其后面的元素 $a_{i+1}\sim a_n$ 都要向上移动一个位置，共移动了 $n-i$ 个元素。

由于 i 的取值范围为：$1\leqslant i\leqslant n$，即有 n 个位置可以插入。设在第 i 个位置上作删除的概率为 P_i，所以平均移动数据元素的次数：

$$E_{de}=\sum_{i=1}^{n}p_i(n-i)$$

在等概率情况下，即 $p_i=1/n$，则：

$$E_{de}=\sum_{i=1}^{n}p_i(n-i)=\frac{1}{n}\sum_{i=1}^{n+1}(n-i)=\frac{n-1}{2}$$

这说明顺序表上作删除运算时大约需要移动表中一半的元素，显然该算法的时间复杂度为 $O(n)$。

(4)按值查找

线性表中的按值查找是指在线性表中查找与给定值 x 相等的数据元素。在顺序表中完成该运算最简单的方法是：从第一个元素 a_1 起依次和 x 比较，直到找到一个与 x 相等的数据元素，则返回它在顺序表中的存储下标或序号(二者差 1)；或者查遍整个表都没有找到与 x 相等的元素，返回 ERROR。

【算法如下】

```
int Locate_SqList (SqList L, ElemType x) {
   i = 0;
   while(i < = L. length - 1 && L. elem[ i ] ! = x)
     i + + ;
   if (i > L. length - 1)   return ERROR;
   else return i;                          // 返回的是存储位置
}
```

算法 1.4

【性能分析】

本算法的主要运算是比较,显然比较的次数与 x 在表中的位置有关,也与表长有关。当 $a_1 = x$ 时,比较 1 次成功,当 $a_n = x$ 时,比较 n 次成功,按值查找的平均比较次数为 $(n+1)/2$,时间性能为 $O(n)$。

3. 顺序表应用举例

【例 1.1】 已知 $A = (a1, a2, \cdots, am)$,

$B = (b1, b2, \cdots, bn)$

均为顺序表,试编写一个比较 A,B 大小的算法。

【分析】

(1)算法的目标是分析两个表的大小,则算法中不应当破坏原表;

(2)按题意,表的大小指的是"词典次序",则不应当先比较两个表的长度;

(3)算法中的基本操作为:同步比较两个表中相应的数据元素,$A > B$ 函数返回 1;$A = B$ 返回 0;$A < B$ 返回 -1。

【算法如下】

```
int compare(SqList La, SqList Lb) {
    i = 0;
    while (i < La. Length && i < Lb. Length) {
        if (La. elem[i] = = Lb. elem[i]) i + + ;
        else if (La. elem[i] < Lb. elem[i])
                return -1;
            else return 1;
    }
    if ( i > La. length && i > Lb. Length) return 0;
    else if (i > Lb. Length) return 1;
        else return -1;
}
```

算法 1.5

算法的时间性能是 O(La. length + Lb. length)。

1.3 线性表的链式存储结构

由于顺序表的存储特点是用物理上的相邻实现了逻辑上的相邻,它要求用连续的存储单元顺序存储线性表中各元素,因此,对顺序表插入、删除时需要通过移动数据元素来实现,影响了运行效率。本节介绍线性表的链式存储结构,它不需要用地址连续的存储单元来实现,因为它不要求逻辑上相邻的两个数据元素物理上也相邻,它是通过"链"建立起数据元素之间的逻辑关系,因此对线性表的插入、删除不需要移动数据元素。

1.3.1 单链表

1. 链表表示

链表是通过一组任意的存储单元来存储线性表中的数据元素。为建立起数据元素之间的线性关系,对每个数据元素 a_i,除了存放数据元素的自身的信息 a_i 之外,还需要和 a_i 一起存放其后继 a_{i+1} 所在的存储单元的地址,这两部分信息组成一个"结点",结点的结构如图 1-1-3 所示。

图 1-1-3　单链表结点结构

其中，存放数据元素信息的称为数据域，存放其后继地址的称为指针域。因此 n 个元素的线性表通过每个结点的指针域拉成了一个“链”，称之为链表。因为每个结点中只有一个指向后继的指针，所以称其为单链表，如图 1-1-4 所示。

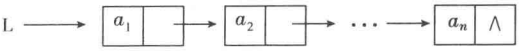

图 1-1-4　单链表示意图

线性表的单链表存储结构的 C 语言描述如下：

```
typedef struct LNode{
    ElemType      data;              // 数据域
    struct LNode  *next;             // 指针域
}LNode, *LinkList;
LinkList L;                          // L 为单链表的头指针
```

通常用“头指针”来标识一个单链表，如单链表 L、单链表 H 等，是指某链表的第一个结点的地址放在了指针变量 L、H 中，头指针为“NULL”则表示一个空表。

2. 单链表上基本运算的实现

(1)建立单链表

●头插法——在链表的头部插入结点建立单链表。

链表与顺序表不同，它是一种动态管理的存储结构，链表中的每个结点占用的存储空间不是预先分配，而是运行时系统根据需求而生成的，因此建立单链表从空表开始，每读入一个数据元素则申请一个结点，然后插在链表的头部。图 1-1-5 展现了线性表：(1,2,3,4,5)在链表的头部插入结点建立链表的过程。因为是在链表的头部插入，读入数据的顺序和线性表中的逻辑顺序是相反的。

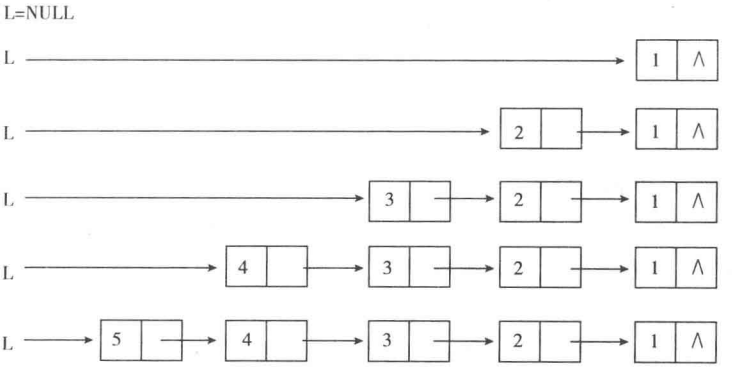

图 1-1-5　头插法建立单链表过程

【算法如下】

```
LinkList  Create_LinkList1 ( ){
    LinkList L = NULL;          // 空表
    LNode  *s;
    int x;                      // 设数据元素的类型为 int
```