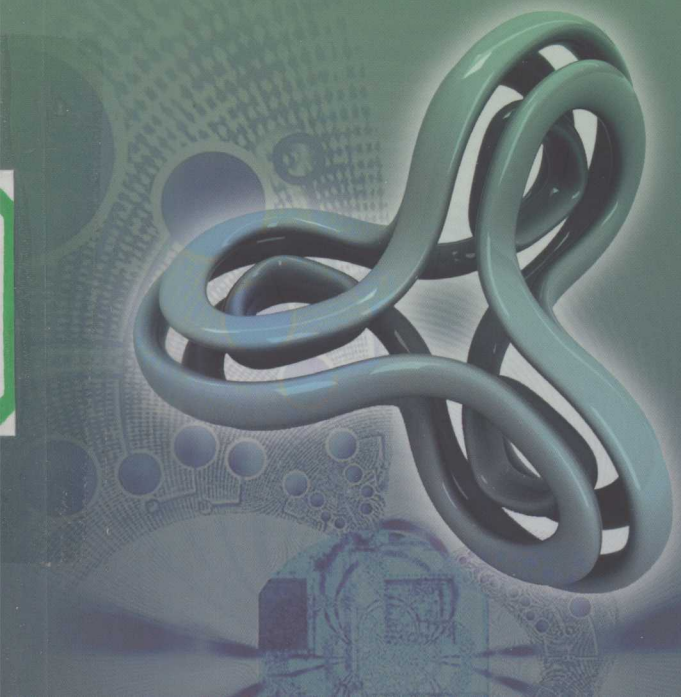
HZ BOOKS
华章教育

计算机学科硕士研究生
入学统一考试课程参考教材

Data Structures
数据结构
考研指导

试题研究编写组



涵盖最新考研大纲
紧扣大纲设计题目
考点解析透彻清楚
资深命题阅卷团队

机械工业出版社
China Machine Press

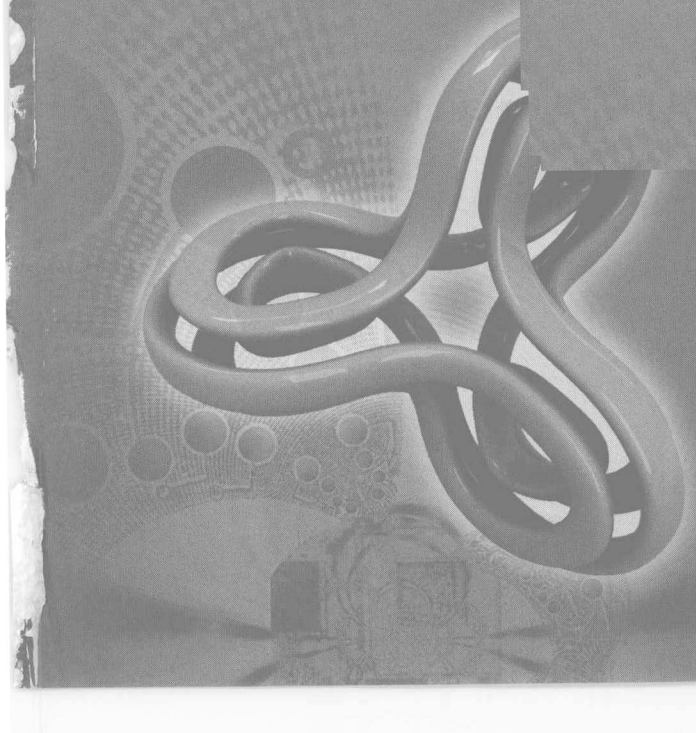
计算机学科硕士研究生
入学统一考试课程参考教材

Data Structures

数据结构

考研指导

试题研究编写组



涵盖最新考研大纲

紧扣大纲设计题目

考点解析透彻清楚

资深命题阅卷团队

机械工业出版社
China Machine Press

本书按线性逻辑、层次逻辑、网状逻辑的顺序讲解数据结构的基本概念，根据学生对新知识学习认知的规律，对每种数据结构从数据的逻辑结构开始，逐渐地引入数据的存储结构和相关的方法，达到深化学生对概念的理解和掌握的目的。另外，本书在对数据结构进行深入研究的基础上，通过分析应用实例以及经典的算法设计方法，更加强调数据结构的应用。

本书适合计算机相关专业的学生用于考研的参考书，也可供本科生学习数据结构课程时参考。

图书在版编目（CIP）数据

数据结构考研指导/试题研究编写组．—北京：机械工业出版社，2009.6
（计算机学科硕士研究生入学统一考试课程参考教材）

ISBN 978-7-111-26771-3

Ⅰ.数…　Ⅱ.试…　Ⅲ.数据结构-研究生-入学考试-自学参考资料　Ⅳ.TP311.12

中国版本图书馆 CIP 数据核字（2009）第 050787 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码 100037）
责任编辑：王　璐
北京慧美印刷有限公司印刷
2009 年 6 月第 1 版第 1 次印刷
184mm×260mm　·20 印张
标准书号：ISBN 978-7-111-26771-3
定　　价：36.00 元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换
本社购书热线：（010）68326294

PREFACE 前言

数据结构是计算机及其相关专业的核心课程，也是全国硕士研究生入学考试计算机专业的必考科目之一。本书由长期坚持在教学一线的教师亲自主笔，在整理多年教学经验、分析考研试题的基础上编写的。书中融汇了数据结构这门课程的特点、难点、知识点和考研的出题重点，在内容的选取上符合计算机专业考研大纲要求，并兼顾学科的广度和深度，提供了丰富的例题和练习题，其中，有很多题目取自部分名校的研究生入学试题真题，并从应试思路上对这些题目进行解析。

本书采用类C语言作为数据结构和算法的描述语言，在内容的取舍上紧扣教育部研究生入学统一考试大纲要求。本书从指导课程教学、学习和考试的角度出发，通过对大量常见经典题型的分析，教授一种数据结构的解题方法、解题规律和解题技巧。这对提高读者分析问题的能力，理解基本要领和理论，开拓解题思路，将会起到良好的效果。主要内容分为6章。第1章是线性表；第2章是栈和队列；第3章是树和二叉树；第4章是图；第5章是查找；第6章是内部排序。各章均由核心考点、例题分析、基础要点总结、习题及解析4部分组成。书中习题及解析部分强调解题思路，注重算法分析。其中的题目全部选自数据结构课程的经典题库和名校考研真题，对其进行详细分析解答，以供读者了解课程考试与考研的深度和模式，进行实战演练。

本书适合参加计算机及相关专业硕士研究生入学考试的学生采用，也可作为计算机类专业或信息类专业的本科教材，还可供从事计算机工程与应用工作的科技工作者参考。

由于作者水平有限，书中存在疏漏与不妥之处，恳请读者批评指正。

编　者

2009年5月

目 录 CONTENTS

CHAPTER1 第1章

线 性 表

线性结构的特点是：在数据元素的非空有限集合中，存在一个唯一的数据元素，该元素称为第一个元素；存在一个唯一的数据元素称为最后一个元素；除第一个元素之外，其余的数据元素均有唯一的直接前驱元素；除最后一个元素之外，其余的数据元素均有唯一的直接后继元素。在线性表这一部分，大纲中首先给出的是有关线性表的定义和基本操作，对于这部分内容我们给出了线性表的定义、线性表的逻辑结构和线性表的基本操作。在这一部分的题目中我们只给出逻辑上的思路和描述，具体存储方式及实现在后面的章节进行分析。

❑1.1 线性表的定义和基本操作

1.1.1 线性表的定义

线性表是具有 $n(n\geq 0)$ 个元素的一个有限序列。线性表中元素的个数 n 定义为线性表的长度，当 $n=0$ 时称为空表，用一对空括号表示；当 $n>0$ 时可表示为 $(a_1, a_2, \cdots, a_n)$，其中 a_1 称为表头元素，a_n 称为表尾元素，a_{i-1} 称为 $a_i(i\geq 2)$ 的直接前驱，a_{i+1} 称为 $a_i(i\leq n-1)$ 的直接后继。

线性表中的数据元素可以是一个数，或一个符号，也可以是一个复杂类型，但同一线性表中的数据元素必须具有相同的属性。

1.1.2 线性表的逻辑结构

线性表的逻辑结构是线性结构，元素之间是 1 对 1 的关系，即除表头元素之外，每个元素（结点）有且只有一个直接前驱，除表尾元素外，每个元素有且只有一个直接后继，当表中只有一个元素 a_1 时，它既没有前驱元素也没有后继元素。

下面给出一些关于线性表的定义和线性表的逻辑结构的习题。

【习题及解析】

【例 1-1】 线性表是具有 n 个（ ）的有限序列（$n\geq 0$）。

A. 表元素　　B. 字符　　C. 数据元素　　D. 数据项　　E. 信息项

【分析】 我们经常在一些参考书上遇到此题。但是，注意区分不同的概念，数据元素是数据的基本单位；数据元素可由若干数据项组成。因此，此题的题干有一定的问题，说法不妥当，选择 C 数据元素或者 D 数据项都是可以的。

【解答】 C。

【例 1-2】 线性表是（ ）。

A. 一个有限序列，可以为空　　B. 一个有限序列，不能为空

C. 一个无限序列，可以为空　　D. 一个无限序列，不能为空

【分析】 根据线性表的定义：线性表是具有 $n(n\geqslant 0)$ 个元素的一个有限序列，当 $n=0$ 时称为空表。

【解答】 A。

【例 1-3】 关于线性表的说法，下面选项正确的是（ ）。

A. 线性表的特点是每个元素都有一个前驱元素和一个后继元素。

B. 线性表是具有 $n(n\geqslant 0)$ 个元素的一个有限序列。

C. 线性表就是顺序存储的表。

D. 线性表只能用顺序存储结构实现。

【分析】 根据线性表的逻辑结构的描述：线性表的逻辑结构是线性结构，元素之间是 1 对 1 的关系，即除表头元素之外，每个元素（结点）有且只有一个直接前驱，除表尾元素外，每个元素有且只有一个直接后继，当表中只有一个元素 a_1 时，它既没有前驱元素也没有后继元素。所以 A 选项是错误的。

根据线性表的定义：线性表是具有 $n(n\geqslant 0)$ 个元素的一个有限序列。因此，B 选项是正确的。

线性表是逻辑结构，可以顺序存储，也可链式存储。所以，C、D 选项都是错误的。

【解答】 B。

1.1.3 线性表的基本操作

以下给出有关线性表的基本操作：

①*InitList*(L)：初始化表。构造一个空的线性表。

②*Length*(L)：求表长。返回线性表 L 的长度，即 L 中数据元素的个数。

③*GetElem*(L, i)：取表中元素。当 $1\leqslant i\leqslant Length(L)$ 时，返回 L 中的第 i 个元素 a_i 的值（或 a_i 的存储位置）；否则返回一个特殊值。

④*PriorElem*(L, x, pre_x)：取元素 x 的直接前驱。设 $x=a_i$，当 $2\leqslant i\leqslant Length(L)$ 时，返回 a_i 的直接前驱 a_{i-1}。

⑤*NextElem*$(L, x, next\_x)$：取元素 x 的直接后继。设 $x=a_i$，当 $1\leqslant i\leqslant Length(L)-1$ 时，返回 a_i 的直接后继 a_{i+1}。

⑥*LocateElem*(L, x)：定位。返回元素 x 在线性表 L 中的位置。若在 L 中存在多个 x，则只返回第一个 x 的位置，若在 L 中不存在 x，则返回 0。

⑦*ListInsert*(L, i, x)：插入元素。在线性表 L 的第 i 个位置上插入元素 x，运算结果使得线性表的长度增加 1。

⑧*ListDelete*(L, i)：删除元素。删除线性表 L 的第 i 个位置上的元素 a_i，此运算的前提应是 $Length(L)\neq 0$，运算结果使得线性表的长度减 1。

⑨*PrintList*(L)：输出线性表。按前后顺序输出线性表 L 的所有元素。

下面给出一些题目有关线性表逻辑上的操作。

【习题及解析】

【例 1-4】 下面关于线性表的叙述中，错误的是哪一个？（ ）

A. 线性表采用顺序存储，必须占用一片连续的存储单元。

B. 线性表采用顺序存储，便于进行插入和删除操作。

C. 线性表采用链接存储，不必占用一片连续的存储单元。

D. 线性表采用链接存储，便于插入和删除操作。

【分析】 线性表是逻辑结构，可以顺序存储，也可链式存储。采用顺序存储时，顺序表是采用数组实现的，必须占用一片连续的存储单元，这是一种随机存取结构，即对表中任一结点都可在 $O(1)$ 时间内直接存取，适宜于静态查找，而要进行插入和删除操作时，则需移动大量结点。因此，选项 A 是正确的。选项 B 是错误的，线性表采用顺序存储，便于查找，而不宜进行插入和删除操作。

采用链接存储时，线性表不必占用一片连续的存储单元。链表不是一种随机存取结构，查找某个结点时，需从头指针开始沿链扫描才能取得，所以不宜做查找；但对插入和删除操作，都只需修改指针，所以链表宜做这种动态的插入和删除操作。因此，选项 C、D 也是正确的。

【解答】 B。

【例 1-5】 简述将两个有序表合成为一个有序表的过程。

【分析】 此题为线性表中的典型题目之一。我们在此只给出逻辑上的算法思想，在后面会给出顺序存储结构和链式存储结构下的算法实现。

【解答】 设有序表 A、B 均为递增序列，要将其合并为一个新的递增序列 C，其算法思想如下：

设 A 表的长度为 n，B 表的长度为 m，则合并之后 C 表的长度应为 $m+n$。

定义三个指针 p、q、r，分别指向 A 表的第一个元素、B 表的第一个元素和 C 表的第一个元素。

当指针 p 不是指向 A 表的最后一个元素，且 q 不是指向 B 表的最后一个元素时，反复执行下面的操作：比较 p、q 所指向的元素的大小，若 p 所指向的元素较小，则将 p 所指向的元素复制到 r 所指向的位置，并将 p、r 均后移一个元素；否则，则将 q 所指向的元素复制到 r 所指向的位置，并将 q、r 后移一个元素。

在实际题目中，可能会对本题的条件加以限制，如在顺序存储结构中，可能要求合并后的结果不另设新表存储，而是存储在表 A 或表 B 中；再比如题干中的表 A 和表 B 可能是两个循环链表。此时，我们只要将上述方法做相应调整即可达到题目的要求。

在大纲中，接下来要考查线性表的实现，下面首先讲述线性表的顺序存储结构以及线性表的基本运算的实现。在这一部分的题目，我们给出算法思想以及具体算法的实现。

1.2 线性表的实现

1.2.1 线性表顺序存储结构

线性表的**顺序存储**指的是用一组地址连续的存储单元依次存储线性表的数据元素。顺序存储的特点是逻辑结构中相邻的结点在存储结构中仍然相邻，而且可以随机存取。

设已知线性表的第一个数据元素 a_1 的存储位置，且线性表中每个数据元素需占用 C 个存储单元，并且所占的第一个单元的存储地址作为数据元素的存储位置，则线性表中的第 i 个数据元

素的存储位置为

$$LOC(a_i) = LOC(a_1) + (i-1) \times C \quad (1 \leqslant i \leqslant n)$$

(1) 顺序表的类型定义

我们在定义中利用数组作为顺序表的存储结构。数组可以是静态的也可以是动态数组，我们常用的是动态数组。

顺序表的类型定义如下：

```
#define MaxSize <顺序表的容量>
typedef struct{
        ElemType data[MaxSize];
        int len;
        }Incode;
```

(2) 顺序表上基本操作的实现

由于C语言中数组的下标是从0开始的，所以在逻辑上所指的“第k个位置”实际上对应的是顺序表的“第$k-1$个位置”。在顺序表上实现线性表的基本运算的函数如下：

```
①InitList(L)                          //初始化表
  void InitList(Incode &L)
  {L.len=0;
  }
②Length(L)                            //求表长
  int Length(Incode L)
  { return L.len;
  }
③GetElem(L,i)                         //取表中元素
  int GetElem(Incode L,int i)
  {
  return L.data[i-1];
  }
④PriorElem(L,x,pre_x)                 //取元素 a_i(a_i=x)的直接前驱
  int PriorElem(Incode L,int x,int pre_x)
  { int i=0;
   while(L.data[i]!=x)
    i++;
   pre_x=L.data[i-1];
   return pre_x;
  }
⑤NextElem(L,x,next_x)                 //取元素 a_i(a_i=x)的直接后继
  int NextElem(Incode L,int x,int next_x)
  { int i=0;
   while(L.data[i]!=x)
    i++;
   next_x=L.data[i+1];
   return next_x;
  }
⑥LocateElem(L,x)                      //定位
  int LocateElem(Incode L,ElemType x)
```

```
{ int i=0;
 While(L.data[i]!=x)                    //查找为x的第1个结点
    i++;
 if(i>L.len)return 0;
 else return i+1;
}
⑦ListInsert(L,i,x)                      //插入元素
int ListInsert(Incode L,int i,ElemType x)
{ int j;
 if(i<1||i>L.len)                       //无效的参数i
   return 0;
 for(j=L.len;j>i;j--)                   //将序号为i的结点及之后的结点后移
   L.data[j]=L.data[j-1];
 L.data[i-1]=x;                         //在序号i处放入x
 L.len++;                               //线性表的长度增加1
 return 1;
}
⑧ListDelete(L,i)                        //删除元素
int ListDelete(Incode L,int i,)
{ int j;
 if(i<1||i>L.len)                       //无效的参数i
    return 0;
 for(j=i;j<L.len;j++)                   //将序号为i的结点及之后的结点前移
    L.data[j-1]=L.data[j];
 L.len--;                               //线性表的长度减1
 return 1;
}
⑨PrintList(L)                           //输出线性表
 void PrintList(Incode L)
{ int i;
 for(i=1;i<=L.len;i++)
    printf("%d",L.data[i-1]);
 printf("\n");
}
```

int LocateElem(*Incode L*, *ElemType x*) 是顺序表的顺序查找算法，其主要思想是：从表的开始位置起，根据给定值 x，逐个与表中各表项的值进行比较。若给定值与某个表项的值相等，则算法报告查找成功的信息并返回该表项的位置；若查遍表中所有的表项，没有任何一个表项满足要求，则算法报告查找不成功的信息并返回一个 -1（也可修改算法，返回新表项应该插入的位置）。

查找算法的时间代价用数据比较次数来衡量。在查找成功的情形下，顺序查找的数据比较次数可做如下分析：若要找的正好是表中第 0 号表项，数据比较次数为 1，这是最好的情况；若要找的是表中最后的 $n-1$ 号表项，数据比较次数为 n（设表的长度为 n），这是最坏的情况。查找第 i 号表项的数据比较次数为 $i+1$，则查找的平均数据比较次数 *ACN*（Average Comparing Number）为：

$$ACN = \sum_{i=0}^{n-1}(i+1) = \frac{1}{n}(1+2+\cdots+n) = \frac{1}{n}\times\frac{(n+1)\times n}{2} = \frac{n+1}{2}$$

即平均要比较 $(n+1)/2$ 个表项。

分析顺序表的插入和删除算法的时间代价主要看循环内的数据移动次数。在插入时，必须从后向前循环，逐个移动 $(n-1)-i+1=n-i$ 个表项。因此，最好的情形是在第 n 个位置追加新表项，移动表项个数为0；最坏情形是在第0个位置插入新表项，移动表项个数为 n；平均数据移动次数 *AMN*(Average Moving Number) 在各表项插入概率相等时为：

$$AMN = \frac{1}{n+1}\sum_{i=0}^{n}(n-i) = \frac{1}{n+1}(n+\cdots+1+0) = \frac{1}{(n+1)}\frac{n(n+1)}{2} = \frac{n}{2}$$

即就整体性能来说，在插入时有 $n+1$ 个插入位置，平均移动 $n/2$ 个表项。

因此，插入算法的平均时间复杂度为 $O(n)$。

在删除时，必须从前向后循环，逐个移动 $(n-1)-(i+1)+1=n-i-1$ 个表项。因此，最好的情形是删除最后的第 $n-1$ 号表项，移动表项个数为0；最坏情形是删去第0号表项，移动表项个数为 $n-1$；平均数据移动次数 *AMN* 在各表项删除概率相等时为：

$$AMN = \frac{1}{n}\sum_{i=0}^{n-1}(n-i-1) = \frac{1}{n}((n-1)+\cdots+1+0) = \frac{1}{n}\frac{(n-1)n}{2} = \frac{n-1}{2}$$

就整体性能来说，在删除时有 n 个删除位置，平均移动 $(n-1)/2$ 个表项。

因此，插入算法的平均时间复杂度为 $O(n)$。

我们来看下面几个习题。

【习题及解析】

【例1-6】 下述哪一条是顺序存储结构的优点？( )

A. 存储密度大　　B. 插入运算方便

C. 删除运算方便　　D. 可方便地用于各种逻辑结构的存储表示

【分析】 顺序表是静态分配的，其存储密度为1；顺序表是采用数组实现的，是一种随机存取结构，即对表中任一结点都可在 $O(1)$ 时间内直接存取，适宜于静态查找，而要进行插入和删除操作时，则需移动大量结点。

【解答】 A。

【例1-7】 将两个各有 n 个元素的有序表归并成一个有序表，其最少的比较次数是( )。

A. n　　B. $2n-1$　　C. $2n$　　D. $n-1$

【分析】 当一个表的最小元素大于另一个表的最大元素时，此时，比较次数为最少，比较次数为 n 次。因此，选项A是正确的，B、C、D错误。

【解答】 A。

【例1-8】 表长为n的顺序存储的线性表，当在任何位置上插入或删除一个元素的概率相等时，插入一个元素所需移动元素的平均个数为(1)，删除一个元素需要移动元素的平均个数为(2)。

A. $(n-1)/2$　　B. n　　C. $n+1$　　D. $n-1$

E. $n/2$　　F. $(n+1)/2$　　G. $(n-2)/2$

【分析】 插入时，必须从后向前循环，逐个移动 $(n-1)-i+1=n-i$ 个表项。因此，最好的情形是在第 n 个位置追加新表项，移动表项个数为0；最坏情形是在第0个位置插入新表项，移动表项个数为 n；平均数据移动次数 *AMN*(Average Moving Number) 在各表项插入概率相等时为：

$$AMN = \frac{1}{n+1}\sum_{i=0}^{n}(n-i) = \frac{1}{n+1}(n+\cdots+1+0) = \frac{1}{(n+1)}\frac{n(n+1)}{2} = \frac{n}{2}$$

即就整体性能来说，在插入时有 $n+1$ 个插入位置，平均移动 $n/2$ 个表项。

在删除时，必须从前向后循环，逐个移动 $(n-1)-(i+1)+1=n-i-1$ 个表项。因此，最好的情形是删除最后的第 $n-1$ 号表项，移动表项个数为0；最坏情形是删去第0号表项，移动表项个数为 $n-1$；平均数据移动次数 *AMN* 在各表项删除概率相等时为：

$$AMN = \frac{1}{n}\sum_{i=0}^{n-1}(n-i-1) = \frac{1}{n}((n-1)+\cdots+1+0) = \frac{1}{n}\frac{(n-1)n}{2} = \frac{n-1}{2}$$

就整体性能来说，在删除时有 n 个删除位置，平均移动 $(n-1)/2$ 个表项。

【解答】 (1) E，(2) A。

【例1-9】 设 A 是一个线性表 $(a_1, a_2, \cdots, a_n)$，若采用顺序存储结构，则在等概率的前提下，平均插入一个元素需要移动的元素个数为多少？若元素插在 a_i 和 a_{i+1} 之间 $(0 \leqslant i \leqslant n-1)$ 的概率为 $\frac{n-i}{n(n+1)/2}$，则平均每插入一个元素所移动的元素的个数又是多少？

【解答】 在等概率的前提下，平均插入一个元素需要移动的元素个数为：

$$(0+1+2+\cdots+n)/(n+1) = \frac{n}{2}$$

若元素插在 a_i 和 a_{i+1} 之间 $(0 \leqslant i \leqslant n-1)$ 的概率为 $\frac{n-i}{n(n+1)/2}$，则平均每插入一个元素所移动的元素的个数为：

$$\sum_{i=0}^{n-1}\frac{(n-i)(n-i)}{n(n+1)/2} = (2n+1)/3$$

【例1-10】 简答题：对于线性表的两种存储结构，如果线性表的总数基本稳定，并且很少进行插入和删除操作，但是要求以最快的速度存取线性表中的元素，则应该选哪种存储结构？试说明理由。

【解答】 应该选线性表的顺序存储结构，因为每个数据元素的存储位置和线性表的起始位置相差一个和数据元素在线性表中的序号成正比的常数，所以只要确定了线性表的起始位置，线性表中任何一个数据元素都可以随机存取。因此，线性表中的顺序存储结构是一种随机存取的存储结构，而链式存储结构是一种顺序存取的存储结构。

【例1-11】 设顺序表 L 中的数据元素递增有序。试编写一算法，将 x 插入到顺序表的适当位置上，以保持线性表的有序性。

【分析】 本题的算法思想是：

①查找 x 在顺序表中的插入位置，即求满足 L.data [i] ≤x＜L.data [i+1] 的 i 值（i 的初值为 L.len-1）；

②将顺序表中的 L.len-i-1 个元素 L.data [i+1] 至 L.data [L.len-1] 后移一个位置；

③将 x 插入到 L.data [i+1] 中且将表长 L.len 加1。

上述算法正确执行的参数条件为：0≤L.len＜MaxSize。

【解答】 算法如下：

```
struct InsertOrderList (Inode &L,ElemType x)
{       //顺序表 L 中的元素依次递增有序,将 x 插入表中合适位置
        if(L.len == MaxSize)
            return 0;
        else {
            i = L.len -1;
            while(i > =0 && x < L.data[i])i --;          //查找 x 的插入位置
              for(j = L.len -1;j > = i +1;j --)
                L.data[j +1] = L.data[j];                //元素后移
            L.data[i +1] = x;                            //插入 x
            L.len ++;                                    //表长加 1
            return 1;
            }
}
```

该算法的时间复杂度为 $O(n)$。

【例 1-12】 已知一个线性表中的元素按元素值非递减有序排列，编写一个函数删除线性表中多余的值相等的元素。

【分析】 本题的算法思想是：由于线性表中的元素按元素值非递减有序排列，值相同的元素必为相邻的元素，因此，依次比较相邻两个元素，若值相等，则删除其中一个，否则继续向后查找，最后返回线性表的新长度。

【解答】 算法如下：

```
int del(Inode L,int len)              //线性表的长度为 len
{ int i =0,j;
 while(i < =len -1)
   if(L.data[i]! =L.data[i +1])   //元素值不等,继续向下查找
     i ++;
 else
    { for(j = i;j < len;j ++)
        L.data[j] = L.data[j +1];   //删除第 i +1 个元素
          len --;                   //表长度减 1
    }
 return len;
}
```

【例 1-13】 编写一函数，将一个线性表 L(有 len 个元素并且任何元素均不为 0) 分拆成两个线性表，使 L 中大于 0 的元素存放在 A 中，小于 0 的元素存放在 B 中。

【分析】 本题的算法思想是：依次遍历 L 中的元素，比较当前的元素值，大于 0 的赋给 A(假设有 p 个元素)，小于 0 的赋给 B(假设有 q 个元素)。

【解答】 算法如下：

```
void ret(Inode L,Inode A,Inode B,int len,int *p,int *q)
{int i;
 *p =0;*q =0;
 for(i =0;i < =len -1;i ++)
    {if(L.data[i] >0)
```

```
            {(*p)++;
             A.data[*p]=L.data[i];
            }
          if(L.data[i]<0)
            {(*q)++;
             B.data[*q]=L.data[i];
            }
        }
    }
```

【例 1-14】 试编写一个函数，以不多于 $3n/2$ 的平均比较次数，在一个有 n 个整数的顺序表 A 中找出具有最大值和最小值的整数。

【分析】 本题的算法思想是：如果在查找出最大值和最小值的元素时各扫描一遍所有元素，则至少要比较 $2n$ 次，为此，使用一趟扫描找出最大和最小值的元素。

【解答】 算法如下：

```
void MaxMin(Inode L,int len){
//在顺序表 L 中一趟扫描,分别找最大值 Max 与最小值 Min
    int Max,Min,i;
    Max=L.data[0];
    Min=L.data[0];
    for(i=1;i<len;i++){
        if(L.data[i]>Max)Max=L.data[i];
        else if(L.data[i]<Min)Min=L.data[i];
    printf("Max=%d,Min=%d\n",Max,Min);
    }
}
```

在这个函数中，最坏情况是线性表 L 中 n 个整数按从大到小非递增排列时，这时（L.data[i]＞Max）条件不成立，这是比较的次数为（$n-1$），另外，每次都要比较 L.data[i]＜Min，同样所花的比较次数为（$n-1$），因此，总的数据比较次数为 $2(n-1)$ 次。

最好情况是线性表 L 中 n 个整数按从小到大递增排列时，这时（L.data[i]＞Max）条件均成立，不会再执行 else 的比较，所以总的数据比较次数为 $n-1$ 次。

数据平均比较次数为 $(2(n-1)+n-1)/2=3n/2-3/2<3n/2$。

所以，该函数的平均比较次数不多于 $3n/2$。

【例 1-15】 编写在顺序表上统计出值为 x 的元素的个数的算法，统计结果由函数值返回。

【解答】 从顺序表上统计出值为 x 的元素的个数的算法：

```
int Count(Inode &L,ElemType x)
{int count=0;
 for(i=0;i<L.len;i++)
   if(L.data[i]==x)     count++;
 renturn count;
}
```

【例 1-16】 试编写一个用顺序存储结构实现的将两个有序表合成一个有序表，合并后的结果不另设新表存储的算法（假设表的容量大于或等于两表元素数目之和）。

【分析】 本题的算法思想是：

①设 A 和 B 均为有序表，合并后的结果仍然放在 A 表中；

②设线性表存储空间的初始分配量为 A 表和 B 表的表长之和；

③合并过程中需要进行元素的比较，可以先从 A 表和 B 表的最后一个元素逐个向前进行比较，从而使得合并后的结果不影响 A 表中原来存放的元素。

线性表的顺序存储结构表示如下：

```
typedef struct{
                ElemType data[MaxSize];
                int len;
                int listsize;
                }Incode;
```

【解答】 算法如下：

```
Status MergeList_Sq(Inode &A,Lnode B)
//将A和B两个有序表合并成为一个有序表,合并后的结果放在A表中
{   int m,n;
    n=A.len;                    //初始时,n为A表的表长
    m=B.len;                    //初始时,m为B表的表长
    while(m>0)
    //从A表和B表的最后一个元素逐个向前进行比较、合并
    if(n==0||A.data[n]<B.data[m])
    //如果A表的元素比较完,或A表的元素小于B表的元素,做如下操作
      {A.data[n+m]=B.data[m];
    //从后向前将B表元素插入到A表n+m位置
       m=m-1;                   //顺序表向前移动m
      }
    else
    //如果B表的元素小于A表的元素或B表中已经无元素,做如下操作
      {A.data[n+m]=A.data[m];
    //从后向前将B表元素插入到A表n+m位置
       n=n-1;                   //顺序表向前移动n
      }
    A.len=A.len+B.len;      //修改A表的表长
  renturn OK;
}
```

【例 1-17】 编写一算法，实现顺序表的就地逆置，即利用原表的存储空间将线性表 $(a_1, a_2, \cdots, a_n)$ 逆置为 $(a_n, a_{n-1}, \cdots, a_1)$。

【分析】 理解就地逆置的含义，即仍要利用原有的存储空间，设置一个变量 q，再从两个方向进行表头表尾的交换。

【解答】 算法如下：

```
void reverse(SqList &A)  //顺序表的就地逆置
{
    int q;
   for(i=1,j=A.length;i<j;i++,j--)
    {
```

```
            q=A.elem[i];
            A.elem[i]=A.elem[j];
            A.elem[j]=q;
        }
    }//逆置
```

【例1-18】 已知线性表（a_1，a_2，…，a_{n-1}）按顺序存储于内存，每个元素都是整数，试设计用最少时间把所有值为负数的元素移动到全部正数值元素前面的算法。

【分析】 算法思想是：从左向右找到正数 *A. data*[*i*]，从右向左找到负数 *A. data*[*j*]，将两者相交换。循环这个过程直到 *i* 大于 *j* 为止。

【解答】 算法如下：

```
void move(Inode A)
{
   int i=0,j=A.len-1,k;
   ElemType temp;
   while(i<=j)
   {while(A.data[i]<=0)i++;
    while(A.data[j]>=0)j--;
    if(i<j)                          //交换
       {temp=A.data[i];
        A.data[i]=A.data[j];
        A.data[j]=temp;
       }
   }
}
```

【例1-19】 线性表（a_1，a_2，a_3，…，a_n）中元素递增有序且按顺序存储于计算机内。要求设计一算法完成：

①用最少时间在表中查找数值为 *x* 的元素。

②若找到将其与后继元素位置相交换。

③若找不到将其插入表中并使表中元素仍递增有序。

【分析】 顺序存储的线性表递增有序，可以顺序查找，也可折半查找。题目要求“用最少的时间在表中查找数值为 *x* 的元素”，这里应使用折半查找方法。

【解答】 算法如下：

```
void SearchExchangeInsert(ElemType a[], ElemType x)
/*a是具有n个元素的递增有序线性表，顺序存储。本算法在表中查找数值为x的元素，如查到则与其后继交换位置；如查不到，则插入表中，且使原表仍递增有序。*/
{low=0;high=n-1;                     //low和high指向线性表下界和上界的下标
 while(low<=high)
  {mid=(low+high)/2;                 //找中间位置
   if(a[mid]==x)break;               //找到x,退出while循环
   else if(a[mid]<x)low=mid+1;       //到中点mid的右半去查
        else high=mid-1;             //到中点mid的左部去查
  }
if(a[mid]==x && mid!=n)              //若最后一个元素与x相等,则不存在与其后继交换的操作
```