全国高等教育自学考试

数据结构自学辅导

罗文劼 张晓莉 王苗 编著

全国高等教育自学考试指定教材辅导书

华中科技大学出版社

全国高等教育自学考试指定教材辅导书

数据结构自学辅导

罗文劼　张晓莉　王　苗　编著

华中科技大学出版社

图书在版编目(CIP)数据

数据结构自学辅导/罗文劼　张晓莉　王苗　编著
武汉:华中科技大学出版社，　2002 年 9 月
ISBN 7-5609-2814-5

Ⅰ. 数…
Ⅱ. ①罗…　②张…　③王…
Ⅲ. 数据结构-高等教育-自学考试-自学参考资料
Ⅳ. TP311.12

数据结构自学辅导　　罗文劼　张晓莉　王　苗　编著

责任编辑:叶见欣　　封面设计:潘　群
责任校对:张兴田　　责任监印:张正林

出版发行:华中科技大学出版社　　武昌喻家山　邮编:430074　电话:(027)87545012

录　　排:华中科技大学出版照排室
印　　刷:华中科技大学出版社沔阳印刷厂

开本:787×1092　1/16　　印张:12.75　　字数:284 000
版次:2002 年 9 月第 1 版　　印次:2002 年 9 月第 1 次印刷　　印数:1—4 000
ISBN 7-5609-2814-5/TP・480　　定价:17.80 元

前　言

“数据结构”是计算机及其应用、计算机信息管理专业的一门重要的专业基础课程。当用计算机来解决实际问题时，就要涉及到数据的表示及数据的处理，而数据的表示及数据的处理正是数据结构课程的主要研究对象，通过这两方面内容的学习，为后续课程，特别是软件方面的课程打下厚实的知识基础，同时也提供了必要的技能训练。因此，“数据结构”课程在计算机及其应用专业中具有举足轻重的地位。然而，数据结构涉及数据的组织、存储以及运算的实现，对初学者来说较为抽象，往往找不到感觉，不知道如何学习这门课，而面对习题就更是无从下手，作者想借编写本辅导教材的机会结合多年讲授本门课程的经验，将各章的知识要点进行归纳和总结，对难以理解的问题进行通俗的讲解和指导，目的是通过对本辅导教材的阅读，对读者理解“数据结构”的内容，求解“数据结构”的习题有一个明确的提高。

本书是全国高等教育自学考试指导委员会指定的计算机及其应用专业（独立本科段）教材《数据结构》（黄刘生主编，经济科学出版社出版）的配套教材，每一章与原教材相对应。在每一章中是这样安排的：(1)首先对本章的重点难点进行了归纳和总结，个别地方给予通俗的讲解；(2)给出了本章各种典型题分析，使读者能了解本章具有代表性的相关考题的形式和难易程度；(3)给出了教材中每章课后练习题中大部分习题的解答，重点题都讲解了解题思路，这也是学习者所希望的，因为作为初学者面对习题往往会束手无策；(4)部分章节还给了一些常见的基本练习题和答案。

本教材中所用到的数据类型均与教材中的一致，以方便学习者阅读。书的最后还给出了两套模拟试卷及其解答。本书可作为参加“数据结构”课程自学考试同学的辅导教材，也可供其他学习“数据结构”课程的大专院校的同学参考。

本书在编写过程中力求概念清晰，表述正确，通俗易懂，便于自学。但由于编者水平有限，书中难免出现错误或不妥之处，恳请读者批评指正，编者不胜感激。编者联系方式（电子信箱）luowenjie@xinhuanet.com。

编　者

2002 年 1 月

目　录

第1章 概 论

本章导读

本章介绍了一些关于数据结构的概念、学习数据结构的意义、描述算法和分析算法的计算方法。

重点提示：★ 有关数据结构的术语及概念。

★ 理解数据结构的内容。

★ 理解逻辑结构、存储结构和数据运算三方面的概念及相互之间的关系。

★ 掌握估算时间复杂度和空间复杂度的度量方法。

1.1 重点难点

1.1.1 名词解释

1. 有关数据的名词。

(1)数据。数据是信息的载体，它能够被计算机识别、存储和加工处理。它是计算机程序加工的原料，应用程序可处理各种各样的数据。

(2)数据元素。它是数据的基本单位，在不同的条件下，又称为元素、结点、顶点、记录等。

(3)数据项。即具有独立含义的最小单位。有些数据元素是由若干个数据项组成的。

(4)数据对象。又称为数据元素类，是具有相同性质的数据元素的集合。

2. 数据类型。即一个值的集合以及在这些值上定义的一组操作的集合。它包括：

(1)原子类型。其值不可分解。如C语言中的整型、实型、字符型等。

(2)结构类型。其值可分解为若干成分。如C语言中的数组、结构等。

3. 抽象数据类型(ADT)。指抽象数据的组织和与之相关的操作。它可以看作是数据的逻辑结构及其在逻辑结构上定义的操作。

4. 数据结构。指数据元素之间的相互关系，即数据的组织形式。它包括三方面的内容：

(1)逻辑结构。它是数据之间的逻辑关系。

(2)存储结构。它是数据元素及其关系在计算机存储器内的表示。

(3)数据运算。它是对数据对象施加的操作。

5. 两类逻辑结构。

(1)线性结构。线性结构的逻辑特点是：若结构为非空集，有且仅有一个开始结点和一个终端结点，并且所有结点都最多只有一个直接前驱和直接后继，如线性表。

(2)非线性结构。非线性结构的逻辑特点是：一个结点可能有多个直接前驱和多个直接后继，如树形结构和图形结构。

6. 四种常见的存储结构。

(1)顺序存储。顺序存储方法是把逻辑上相邻的元素存储在物理位置相邻的存储单元中，由此得到的存储表示称为顺序存储结构。顺序存储结构是一种最基本的存储表示方法，通常借助于程序设计语言中的数组来实现。

(2)链式存储。链式存储方法对逻辑上相邻的元素不要求其物理位置相邻，元素间的逻辑关系通过附设的指针字段来表示，由此得到的存储表示称为链式存储结构，链式存储结构通常借助于程序设计语言中的指针类型来实现。

(3)索引存储方式。它是通过建立索引表存储结点信息的方法，其中索引表一般存储结点的关键字和一个地址信息，可通过该地址找到结点的其他信息。有稠密索引和稀疏索引之分。

(4)散列存储方式。它是根据结点的关键字来确定该结点的存储地址的方法。

1.1.2　算法的描述和分析

1. 算法。

算法　是对特定问题求解步骤的一种描述，是由指令组成的有限序列。其中每一条指令表示一个或多个操作。一个算法应该具有下列特性。

(1)有穷性。即一个算法必须在有穷步之后结束，即必须在有限时间内完成。

(2)确定性。即算法的每一步必须有确切的定义，无二义性。算法执行时，对应相同的输入仅有唯一的一条执行路径。

(3)可行性。即算法中的每一步都可以通过已经实现的基本运算的有限次执行得以实现。

(4)输入。一个算法具有零个或多个输入，这些输入取自特定的数据对象集合。

(5)输出。即一个算法具有一个或多个输出，这些输出同输入之间存在着某种特定的关系。

2. 对算法的设计要求。

(1)正确。即算法的执行结果应当满足预先规定的功能和性能要求。

(2)可读。即一个算法应当思路清晰、层次分明、简单明了、易读易懂。

(3)健壮。即当输入不合法数据时，算法应能作适当处理，不致于引起严重后果。

(4)高效。即算法应有效地使用存储空间和有较高的时间效率。

3. 算法的性能分析与度量。

(1)时间复杂度。①某算法的时间复杂度是执行该算法所耗费的时间。通常某算法的时间复杂度是问题规模 n 的函数 T(n)。②大 O 记法表示算法的渐进时间复杂度。③常见的渐进时间复杂度有：

$$O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3) < O(2^n)$$

(2)空间复杂度。一个算法的空间复杂度是指该算法所耗费的存储空间。它通常也是问题规模 n 的函数T(n)。

1.2　典型题分析

题型 1：选择题。

(1) 数据结构是一门研究非数值计算程序设计中计算机的(　①)以及它们之间的(　②)和运算等的学科。

①A. 操作对象　　B. 计算方法　　C. 逻辑存储　　D. 数据映像

②A. 结构　　B. 关系　　C. 运算　　D. 算法

(2) 在数据结构中,从逻辑上可以把数据结构分为(　　)。

A. 动态结构和静态结构　　B. 紧凑结构和非紧凑结构

C. 线性结构和非线性结构　　D. 内部结构和外部结构

答案(1)　①A、②B;(2)C.

题型 2:判断题。

(1) 顺序存储方式只能用于线性结构,不能用于非线性结构。(　　)

(2) 基于某种逻辑结构之上的运算,其实现是唯一的。(　　)

答案(1) ×。顺序存储方式能用于存储非线性结构。

(2) ×。基于某种逻辑结构,其存储结构不是唯一的,因此运算也就不唯一。

题型 3:填空题。

(1) 线性结构中元素的关系是(　①);树形结构中元素的关系是(　②);图形中元素的关系是(　③)。

(2) 算法的 5 个重要特性是_____、_____、_____、_____、_____。

答案(1) ①一对一,② 一对多,③多对多。(2) 有穷性、确定性、可行性、输入、输出。

1.3 课后习题选解

1.1 (略)

1.2 试举一个数据结构的例子,叙述其逻辑结构,存储结构,运算这三个方面的内容。

解答 (1)以线性表为例。

线性表的逻辑结构:在数据元素非空有限集中存在唯一的一个被称为"第一个"的数据元素。存在唯一的一个被称为"最后一个"的数据元素,除第一个数据元素外,集合中每个数据元素均只有一个直接前驱,除最后一个数据元素外,集合中每个数据元素均只有一个直接后继,因此线性表是线性结构。

线性表的存储结构:用顺序存储的方法存储线性表中的元素及元素之间的线性关系,即在内存中用一片连续的存储区域存储线性表的数据元素,这样使逻辑上相邻的数据元素在物理存储时也相邻。

线性表的基本运算有:①置空表,②求长度,③取结点,④定位,⑤插入,⑥删除。

1.3 (略)

1.4 设三个函数 f,g,h 分别为:

$$f(n) = 100n^3 + n^2 + 1000$$

$$g(n) = 25n^3 + 5000n^2$$

$$h(n) = n^{1.5} + 5000 * n * \log_2 n$$

请判断下列关系式是否成立:

①$f(n)=O(g(n))$　②$g(n)=O(f(n))$　③$h(n)=O(n^{1.5})$　④$h(n)=O(n\log_2 n)$

解答 ①、②、③成立。

1.5 设有两个算法在同一机器上运行,其执行时间分别为 $100n^2$ 和 2^n,要使前者快于后

者，n 至少要多大？

解答　要使 $100n^2<2^n$，则需要 $n>=15$.

1.6　设 n 为正整数，利用大"O"记号，将下列程序段的执行时间表示为 n 的函数。

①
```
i=1;k=0;
while(i<n){
k=k+10*i; i++;
}
```

②
```
i=0;k=0;
do{
    k=k+10*i; i++;
  }while (i<n);
```

③
```
i=1;j=0;
while (i+j<=n){
if(i>j) j++;
else i++;
}
```

④
```
x=n; /*n>1*/
while (x>=(y+1)*(y+1))
    y++;
```

⑤
```
x=91;y=100;
while(y>0)
    if(x>100){ x=x-10; y--;}
    else x++;
```

解答　①$T(n)=O(n)$；②$T(n)=O(n)$；③$T(n)=O(n)$

④$T(n)=O(n)$；⑤$T(n)=O(1)$.

1.7　算法的时间复杂度仅与问题的规模相关吗？

解答　一个用高级程序语言编写的程序在计算机上运行时所消耗的时间取决于以下因素：

① 与算法选用的策略有关；

② 与问题的规模有关；

③ 与计算机的软、硬件环境有关。

1.8　按增长率由小至大的顺序排列下列各函数：

$$2^n,(3/2)^n,(2/3)^n,n^n,\sqrt{n},n!,2^n,\lg n,n^{\lg n},n^{3/2}。$$

解答　$(2/3)^n<\lg n<\sqrt{n}<n^{3/2}<n^{\lg n}<(3/2)^n<2^n<n!<n^n$.

1.9　有时为比较两个同数量级算法的优劣，须突出主项的常数因子，而将低次项用大"O"记号表示。例如，设

$$T_1(n)=1.39n\lg n+100n+256=1.39n\lg n+O(n)$$

$$T_2(n)=2.0n\lg n-2n=2.0n\lg n+O(n)$$

这两个式子表示当 n 足够大时 $T_1(n)$优于 $T_2(n)$，因为前者的常数因子小于后者。请用此方法表示下列函数，并指出当 n 足够大时，哪一个较优，哪一个较劣？

①$T_1(n)=5n^2-3n+60\lg n$　　②$T_2(n)=3n^2+1000n+3\lg n$

③$T_3(n)=8n^2+3\lg n$　　④$T_4(n)=1.5n^2+6000n\lg n$

解答　①$T_1(n)=5n^2+O(n)$；　②$T_2(n)=3n^2+O(n)$；

③$T_3(n)=8n^2+O(\lg n)$；　④$T_4(n)=1.5n^2+O(n\lg n)$.

当 n 足够大时，$T_4(n)$较优，$T_3(n)$较劣。

第2章 线 性 表

本章导读

线性表是一种最基本、最常用的数据结构，它有两种存储结构——顺序表和链表。本章介绍了线性表的定义、表示和基本运算的实现。重点讨论了线性表的存储结构，以及在顺序、链式两种存储结构上实现的基本运算。

重点提示：★ 线性表的逻辑结构特征。

★ 线性表的顺序存储和链式存储两种存储结构的特点。

★ 在两种存储结构下的基本操作的实现。

2.1 重点难点

2.1.1 名词解释

(1)线性表。线性表是具有相同数据类型的 n(n≥0)个数据元素的有限序列，通常记为：$(a_1,a_2,\cdots,a_n)$，其中 n 为表长，n=0 时称为空表。

要点：一种逻辑结构，其数据元素属于相同数据类型，它们之间的关系是线性关系。

(2)顺序表。顺序表是顺序存储的线性表。

要点：按线性表中的元素的逻辑顺序依次存放在地址连续的存储单元里，其存储特点：用物理上的相邻实现逻辑上的相邻。

(3)链表。链表是用链式存储的线性表。

要点：链表是通过每个结点的链域将线性表的 n 个结点按其逻辑顺序链接在一起的，对每个结点的地址是否连续没有要求。

(4)单链表。单链表中每个结点除了数据域外还有一个指向其后继的指针域。

通常将每个元素的值和其直接后继的地址作为一个结点，通过每个结点中指向后继的指针表示线性表的逻辑结构。

(5)头指针。它是一个指针变量，里面存放的是链表中首结点的地址，并以此来标识一个链表。如链表 H，链表 L 等，表示链表中第一个结点的地址存放在 H、L 中。通常用头指针来唯一标识一个链表。

(6)头结点。指附加在第一个元素结点之前的一个结点，头指针指向头结点。当该链表表示一个非空的线性表时，头结点的指针域指向第一个元素结点；为空表时，该指针域为空。

(7)头结点的作用。其作用有两个：一是使对空表和非空表的处理得到统一；二是在链表的第一个位置上的操作和在其他位置上的操作一致，无须特殊处理。

2.1.2　线性表的顺序存储

1. 顺序表。顺序存储的线性表称为顺序表。

其特点是：用一组地址连续的存储单元来依次存放线性结点，因此结点的逻辑结构和物理次序一致（这是顺序存储的核心所在）。

具体实现：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是再合适不过的了。考虑到线性表的运算有插入、删除等，即表长是可变的，因此，数组的容量需设计得足够大，设用：data[ListSize]来表示数组的容量，其中 ListSize 是一个根据实际问题定义的足够大的整数，线性表中的数据从 data[0] 开始依次顺序存放，但当前线性表中的实际元素个数可能未达到 ListSize 多个，因此，需用一个变量 length 记录当前线性表中实际元素的个数，因此，表空时 length＝0。

这种存储思想的具体实现可以是多样的。

方法一　可以用一个数组和来表示长度的变量，共同完成上述思想，如：

```
DataType  data[ListSize];
int  length
```

这样表示的顺序表如图 2.1 所示。数据元素分别存放在 data[0]到 data[length－1]中，（当然也可以存放在 data[1]到 data[length]中）。

这样使用简单方便，但有时不便管理。

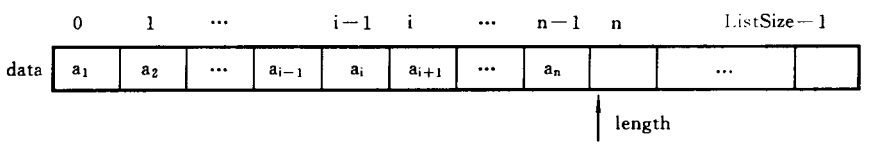

图 2.1　线性表的顺序存储示意图

方法二　从结构性上考虑，通常将 data 和 length 封装成一个结构，作为顺序表的类型：

```
typedef struct
  {  DataType data[ListSize];
     int  length;
} SeqList;
```

定义一个顺序表变量：SeqList　L ；

这样表示的线性表如图 2.2(a) 所示。表长＝L. length，线性表中的数据元素 a_1 至 a_n 分别存放在 L. data[0]至 L. data[L. length－1]中。

方法三　由于书中的算法用 C 语言描述，根据 C 语言中的一些规则，有时定义一个指向 SeqList 类型的指针更为方便：

```
SeqList      * L
```

L 是一个指针变量，线性表的存储空间通过 malloc(sizeof(SeqList)) 操作来获得。

L 中存放的是顺序表的地址，这样表示的线性表如图 2.2(b)所示。表长表示为（＊L). length 或 L－>length，线性表中数据元素的存储空间为：

```
L->data[0] ~ L->data[L->length-1]
```

读者通过上述介绍的几种表示方式，进一步体会顺序存储的“理念”，做题时根据题意灵活掌握，在读写算法时注意相关数据结构的类型说明。

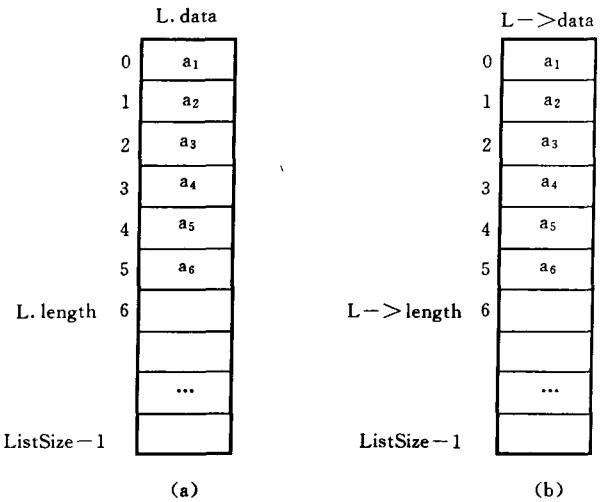

图 2.2 线性表的顺序存储示意图

2.顺序表的优缺点。

优点 1 顺序表是由地址连续的向量实现的，因此，具有按序号随机访问的特点。设 a_1 的存储地址为 $Loc(a_1)$，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：

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

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 个数据元素的地址来，这就是顺序表具有按数据元素的序号随机存取的特点。

优点 2 存储密度高。

缺点 1 作插入和删除运算时，平均需移动大约表中一半元素。

缺点 2 顺序表的存储空间是静态分配的，在程序执行之前必须明确规定它的存储规模，因此，分配不足会造成溢出，分配过大，则可能造成存储单元的浪费。

2.1.3 链表

1.单链表。

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的，那么怎样表示出数据元素之间的线性关系呢？为建立起数据元素之间的线性关系，对于每个数据元素 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 2.3 所示，每个元素都如此。因此 n 个元素的线性表通过每个结点的指针域连成了一个“链子”，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。

链表是由一个个结点构成的，结点定义如下：

```
typedef struct node
        {  DataType data;
```

```
    struct node  * next;
} LinkNode, * LinkList;
```

LinkNode 是结点的类型，* LinkList 是指向 LinkNode 结点的指针类型。

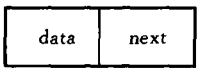

图 2.3　单链表结点结构

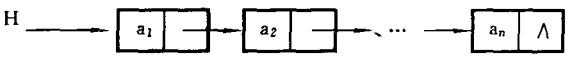

图 2.4　单链表

作为线性表的一种存储结构，我们关心的是结点间的逻辑结构，而对每个结点的实际地址并不关心，所以通常的单链表用图 2.4 所示的形式表示。

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. 单循环链表。

对于单链表而言，最后一个结点的指针域是空指针，如果将该链表的头指针置入该指针域，则链表头尾结点相连，就构成了单循环链表。

对于单链表，只能从头结点开始遍历整个链表，而对于单循环链表，则可以从表中任意结点开始遍历整个链表，不仅如此，有时对链表常做的操作是在表尾、表头进行，此时可以改变一下链表的标识方法，不用头指针而用一个指向尾结点的指针 R 来标识，这可以使得操作效率得以提高。

3. 双循环链表。

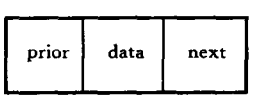

图 2.5　结点的结构

以上讨论的单链表的结点中只有一个指向其后继结点的指针域 next，因此，若已知某结点的指针为 p，其后继结点的指针则为 p－＞next，而要找其前驱，则只能从该链表的头指针开始，顺着各结点的 next 域进行，也就是说，找后继的时间性能是 O(1)，找前驱的时间性能是 O(n)，如果希望找前驱的时间性能也达到 O(1)，则只能付出空间的代价：每个结点再加一个指向前驱的指针域，结点的结构变为如图 2.5 所示，用这种结点组成的链表称为双向链表。

双向链表结点及指针类型定义如下：

```
typedef struct dlistnode
{ DataType data;
  struct dlistnode  * prior, * next;
}DListNode, * DLinkList;
```

和单链表类似，双向链表通常也是用头指针标识。

通过双向链表中某结点的指针 p 即可以直接得到它的后继结点的指针 p－＞next，也可以直接得到它的前驱结点的指针 p－＞prior。这样在有些操作中需要找前驱时，则勿需再用循环的方法。

设 p 指向双循环链表中的某一结点，即 p 是该结点的指针，则 p－＞prior－＞next 表示的是 * p 所指结点之前驱结点的后继结点的指针，即与 p 相等；类似，p－＞next－＞prior 表示的是 * p 所指结点之后继结点的前驱结点的指针，也与 p 相等，所以有以下等式：

$$p->prior->next = p = p->next->prior$$

4. 链式存储的优缺点。

要点：链式存储的优缺点与顺序存储互补。

优点1 作插入和删除运算时，只须改变指针，不需移动数据。

优点2 不需要事先分配空间，便于表的扩充。

缺点1 存储密度降低，因为每个结点中除了存放数据元素的值外还有一个指针域。

缺点2 不具有按序号随机访问第i个元素的特点，必须通过标识链表的头指针(或尾指针)“顺藤摸瓜”才能找到第i个元素。

2.1.4 线性表的基本运算

1. 基于顺序表的运算。

要点：顺序表中的常做的运算有插入、删除、合并、查找等，做这些运算时，要掌握一个原则：时刻体现顺序存储的思想。

(1)在顺序表中插入元素。

①插入元素时，检查顺序表是否已满，满则不能做插入。

②根据具体问题确定插入位置。

③移动有关元素，以便为待插入的元素让出位置来。

④将元素插入。

⑤修改表长。

(2)在顺序表中删除元素。

①检查顺序表是否已空，空则不能做删除。

②根据具体问题确定删除元素的位置。

③删除时，将其后面的有关元素移动，“挤掉”被删除的元素。

④修改表长。

结论： 顺序表中插入一个数据元素平均需要移动$(n+1)/2$个元素。具体到某一次的插入时，移动数据的个数与表长和插入位置有关。

顺序表中删除一个数据元素需要平均移动$n/2$个元素。具体到某一次的删除时，移动数据的个数与表长和删除元素的位置有关。

2. 基于链表的运算。

链表中的操作最好用图示的方法进行，以便清楚指针的变化情况。链表操作过程中，主要的是指针的变化，因此必须清楚指针和动态结点的问题。

(1)头结点。 在对不带头结点的单链表进行操作时，对第一个结点的处理和其他结点是不同的，如：要在单链表的尾部插入结点建立单链表，其算法思路：

在初始状态，头指针 H=NULL，尾指针 r=NULL；按线性表中元素的顺序依次读入数据元素，不是结束标志时，申请结点，将新结点插入到 r 所指结点的后面，然后 r 指向新结点(但第一个结点有所不同，读者注意下面算法中的有关部分)，如图2.6所示。算法如下：

```
LinkList Creat_LinkList( )
  { LinkList L=NULL;
    ListNode *s, *r=NULL;
```

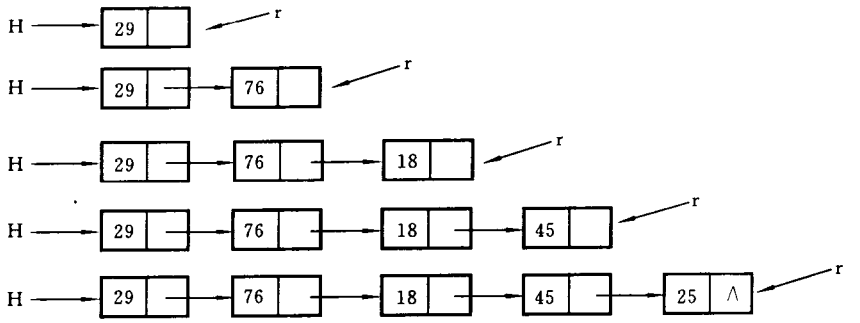

图 2.6　在尾部插入建立单链表

```
    int x;          /* 设数据元素的类型为 int */
    scanf("%d",&x);
    while (x! =flag)  /* flag 表示结束标志,可依据实际情况取值 */
       { s=malloc(sizeof(ListNode));        s->data=x;
         if (L==NULL) L=s; /* 第一个结点的处理 */
         else r->next=s; /* 其他结点的处理 */
         r=s; /* r 指向新的尾结点 */
         scanf("%d",&x);
       }
    if ( r! =NULL) r->next=NULL; /* 对于非空表,最后结点的指针域放空指针 */
    return L;
}
```

可见,在空表的基础上插入结点,由于在第一个结点加入时链表为空,它没有直接前驱结点,它的地址就是整个链表的指针,需要放在链表的头指针变量中;而其他结点有直接前驱结点,其地址放入到直接前驱结点的指针域中。"第一个结点"的问题在很多操作中都会遇到,如在链表中插入结点时,将结点插在第一个位置和其他位置是不同的,在链表中删除结点时,删除第一个结点和其他结点的处理也是不同的,等等,为了方便操作,有时在链表的头部加入一个"头结点",头结点的类型与数据结点一致,标识链表的头指针变量 L 中存放该结点的地址,这样即使是空表,头指针变量 L 也不为空了。头结点的加入使得"第一个结点"的问题不再存在,也使得"空表"和"非空表"的处理得以统一。

头结点的加入完全是为了运算的方便,它的数据域无定义,指针域中存放的是第一个数据结点的地址,空表时为空。

(2) 插入 。

①后插结点　设 p 指向单链表中某结点,s 指向待插入的值为 x 的新结点,将 *s 插入到 *p的后面,插入示意图如图 2.7 所示。操作如下:

```
s->next=p->next;
p->next=s;
```

要点：两个指针的操作顺序不能交换。

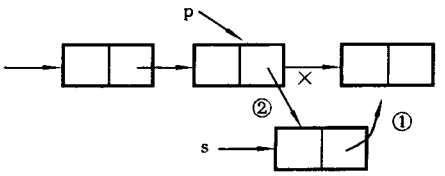

图 2.7 在 *p 之后插入 *s

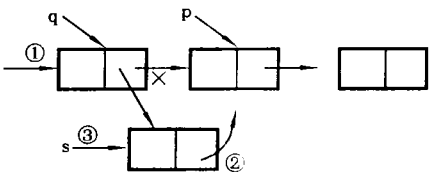

图 2.8 在 *p 之前插入 *s

② 前插结点 设 p 指向链表中某结点，s 指向待插入的值为 x 的新结点，将 *s 插入到 *p的前面，插入示意图如图 2.8 所示，与后插不同的是：首先要找到 *p 的前驱 *q，然后再完成在 *q 之后插入 *s，设单链表头指针为 L，操作如下：

```
q=L;
while (q->next! =p)
    q=q->next;          /* 找 *p 的直接前驱 */
s->next=q->next;
q->next=s;
```

后插操作的时间复杂度为 O(1)，前插操作因为要找 *p 的前驱，时间性能为 O(n)；其实我们关心的更是数据元素之间的逻辑关系，所以仍然可以将 *s 插入到 *p 的后面，然后将 p->data 与 s->data交换即可，这样既满足了逻辑关系，也能使得时间复杂性为 O(1)。

由此得知：在单链表中插入一个结点必须知道其前驱结点。

③ 在单链表第 i 个元素之前插入元素 x。

算法思路：a. 从头指针开始找到第 i-1 个元素结点，若存在，则继续 b. 步骤，否则结束。

b. 申请、填装新结点。

c. 将新结点插入，结束。

(3) 删除。

① 删除结点。

a. 删除 *p 的后继结点(假设存在)，则可以直接完成：

```
s=p->next;          p->next=s->next;
free(s);
```

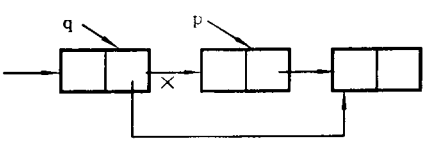

图 2.9 删除 *p

该操作的时间复杂度为 O(1) 。

b. 若要删除 *p，即 p 所指结点，则由图 2.9 可见，首先要找到 *p 的前驱结点 *q，然后完成指针的操作即可。指针的操作由下列语句实现：

```
q->next=p->next;
free(p);
```

显然找 *p 前驱的时间复杂度为 O(n)。

操作示意图如图 2.9 所示。

当 *p 有后继时，对 *p 的删除可以这样处理：把 *p 后继结点的值存储到结点 *p 中，再