21世纪应用型人才培养系列教材

SHU JU JIE GOU

数据结构

王寿福 徐 炜 编著

高等教育出版社
HIGHER EDUCATION PRESS

21世纪应用型人才培养系列教

数据结构

王寿福　　徐　炜　编著

高等教育出版社

内容提要

本书是应用型人才培养系列教材之一，是面向21世纪课程教材。该系列教材根据应用型人才培养的教学基本要求，并参照有关行业最新颁发的职业鉴定规范及高级工等级标准编写。本书主要介绍了数据结构的基本概念、线性表、栈和队列、树、图等常用数据结构以及查找和排序等相关知识。为了配合教学，本书配备了11套模拟试题。本书适合作为高等职业学校、部分本科院校的计算机及相关专业教学用书，也可作为中高级职业资格与就业培训用书。

图书在版编目(CIP)数据

数据结构/王寿福，徐炜编著. —北京：高等教育出版社，2003.6
（高等职业教育系列教材）
ISBN 7-04-012045-3

Ⅰ. 数… Ⅱ. ①王… ②徐… Ⅲ. 数据结构—高等学校：技术学校—教材 Ⅳ. TP311.12

中国版本图书馆CIP数据核字(2003)第041608号

责任编辑 司马镭 **封面设计** 吴 昊 **责任印制** 潘文瑞

书　　名 数据结构
编　　著 王寿福 徐 炜

出版发行	高等教育出版社	**购书热线**	010-64054588
社　　址	北京市西城区德外大街4号		021-56964871
邮政编码	100011	**免费咨询**	800-810-0598
总　　机	010-82028899	**网　　址**	http://www.hep.edu.cn
传　　真	021-56965341		http://www.hep.com.cn
			http://www.hepsh.com
排　　版	南京理工排版校对公司		
印　　刷	江苏省宜兴市德胜印刷有限公司		
开　　本	787×1092 1/16	**版　　次**	2003年6月第1版
印　　张	13.75	**印　　次**	2003年6月第1次
字　　数	326 000	**定　　价**	18.50元

凡购买高等教育出版社图书，如有缺页、倒页、脱页等质量问题，请在所购图书销售部门联系调换。

前　言

数据结构是计算机及相关专业的一门重要的专业基础课程。目前，国内有关数据结构的教材很多，但在内容的选取、算法的编写、习题的安排等方面能真正满足当前应用型人才培养需要的还不多。

本教材着重体现了应用型人才培养的特点，本着理论够用、强调实用的原则，充分体现“培养适应生产、建设、管理、服务第一线需要的高等技术应用型专门人才”的培养目标以及应用型人才教育应知识面宽、基本理论和原理知识适度、加强实践技能培养等要求，在编写该教材时，对传统数据结构课程中的部分内容作了适度的调整，对一些不常用的数据结构和算法作了简化或省略，对一些常用的数据结构和算法作了详细的介绍。为了帮助部分读者自学，编者对算法的重要步骤还作了详尽的注释。

本教材采用流行的C和类C语言描述算法，主要介绍了数据结构的基本概念、线性表、栈、队列、树、图、查找和排序等有关知识，并根据读者的实际需要，在本书的最后附加了数据结构模拟试题并提供了相应的答案。

本教材知识点布局合理，概念表达准确、清晰、简洁，内容由浅入深，既体现理论，又强调实践环节，既利于教师教学，又利于学生自学。

本教材建议90学时，不同地区、院校和专业可根据具体情况作适度调整或有选择地讲授。

本教材适合作为高等职业院校、高等专科学校及部分本科院校的计算机专业或信息类相关专业的教材，也可作为相关专业学生自学考试教材或计算机应用人员和工程技术人员的自学参考书。

本教材第1、2、3、4、8章及附录由王寿福撰写；第5、6、7章由徐炜撰写。全书由王寿福统稿。苏州大学计算机科学与技术系徐汀荣教授审阅了全书并提出了许多宝贵的修改意见。

由于时间仓促和编者水平有限，书中肯定存在缺点和不足之处，恳请各位专家、老师和同学提出宝贵的意见。

编　者

2003年5月

目　录

第1章 绪 论

自第一台电子计算机问世以来，计算机科学与技术得到了飞速的发展，与此同时，计算机的应用范围也从最初的科学计算逐步拓展到人类活动的所有领域，计算机处理的对象也从最初的数值和字符，扩大到图像、图形、声音等多媒体数据，数据的结构也愈来愈复杂。因此，要开发出一种性能良好的软件，不仅要根据实际需要选择一种合适的软件工具，更重要的是要研究数据的不同结构和组织方法，进而设计进行数据处理的算法。这就要求加强对数据结构这门课程的学习。

本章将介绍数据结构研究的对象、基本概念和术语，算法的概念及描述方法(类C语言描述)，数据类型以及抽象数据类型，并概述数据结构的发展概况及其在计算机科学中的地位。

1.1 数据结构的地位

1.1.1 数据结构的概念

一般来说，用计算机解决一个实际问题时，先对具体问题抽象，建立起实际问题的求解模型，然后设计出相应的算法，编写程序并上机调试，最终使实际问题得到解决。当人们用计算机处理的是数值计算问题时，所用的求解模型可以用数学方程描述，所涉及的运算对象一般是整型、实型或逻辑型等一些简单的数据类型。因此，程序设计者的主要精力是集中于程序设计的技巧上，而不是数据的组织和存储上。但随着计算机应用领域的不断扩大，计算机处理的对象更多的是非数值计算问题，如网上资料查询、学生学籍管理、交通道路规划、博弈游戏等问题，它们的数学模型无法用数学方程来进行描述，此时就必须建立相应的数据结构来进行描述：分析问题中所用到的数据是如何组织的，研究数据之间的关系如何，进而为解决这些问题设计出合适的数据结构。

单位要对所有职工的基本情况进行管理，必须要经常了解职工的各种信息(包括职工的编号、姓名、性别以及月收入等情况)，并对整个单位的情况作统计、汇总等工作。如何利用计算机来辅助管理这些信息呢？显然，应先考虑对整个单位中每个职工的信息进行有效的组织和将这些信息存入计算机并利用计算机进行有效处理的方法。

一般可以建立一个花名册对每个职工的信息进行编排，即组织这些信息(表1-1)。花名册中每个职工的信息可由编号、姓名、性别、年龄、月收入等项目组成，占表的一行。这时候整个花名册就构成了一个数据结构，或者称整个花名册就是一个数据结构，而把表中的每一行看作一个结点或一个元素、一个记录，它由编号、姓名、性别、年龄、月收入等项目组成，从而整张表构成了一个关于职工花名册的数学模型，此时的花名册就是一张所谓的线性表，此时计算机就可以按某个特定要求对其进行操作。表中的结点和结点之间是一种简单的线

性关系。表中有且仅有一个结点为表头(开始结点),有且仅有一个结点为表尾(终端结点);对表中任一结点,与它相邻且在它前面的结点最多只有一个,与它相邻且在它后面的结点也最多只有一个,这就是上述花名册表中的逻辑结构。当考虑上述花名册表中的数据用计算机进行计算或处理时,就先要涉及到这些结点如何在计算机中的存储表示的问题,也就是存储结构的问题。对于这样的花名册表,有可能是调入一些职工必须增加新的结点,而某些职工调离时又必须将这些相应结点从表中删除掉。究竟如何进行插入、删除、修改、查找,这就是数据的运算问题,只有弄清这些问题后,才能有效地使用花名册表这个数据结构,有效地解决该单位职工基本信息的计算机辅助管理的问题。

表 1-1 职工花名册表

编 号	姓 名	性 别	年 龄	月收入
1	陈琳	女	20	600
2	王晨	男	43	980
3	张瑜	女	30	700
4	李倩	女	35	800
……	……	……	……	……

若要建立一个旅游交通网络咨询系统,用来回答游客提出的各种问题,就必须对各个城市的情况进行研究,对这些城市所代表的信息进行分析、组织。此时就可采用一种称之为图的结构来表示实际的交通网络,如图 1-1 所示,图中结点表示城市,边表示城市间交通联系,这个交通网络图此时就表示一个数据结构。

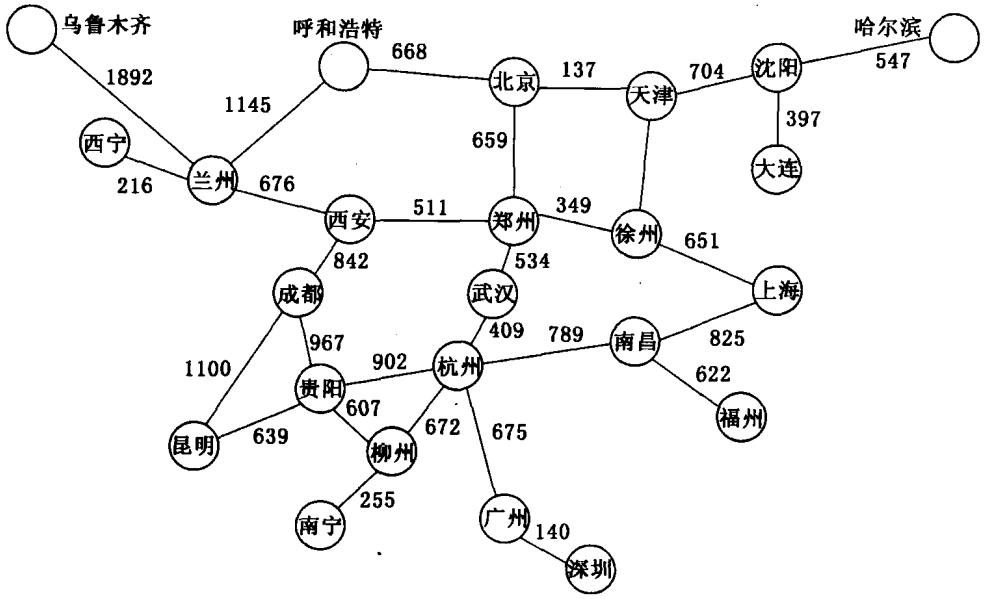

图 1-1 交通图

构造出这张交通图后,要利用计算机进行处理,还得先讨论如何将这张图存入计算机中,这就必须要涉及到所谓图的存储结构问题。也就是说要对图中结点本身的值以及每个

结点之间的可能的关系都存入计算机。然后再利用计算机进行操作处理，这就是所谓对图的各种运算。如一位旅客要从 A 城到 B 城，他希望选择一条旅途中转次数最少的路线，假设图中每一站都需要换车，这个问题反映到图上就是要找一条从顶点 A 到顶点 B 包含边的数目最少的路径，也就是所谓利用图进行的最短路径的运算问题。还有一些诸如关键路径等问题都可以认为是对图这个数据结构所作的运算。

从以上两个例子可见，可描述这类的非数值计算问题的数学模型，单靠数学方程是无法满足的，它涉及到一些诸如表、图还有树之类的数据结构，而这些就是数据结构的研究对象。所以数据结构实际上是一门研究非数值计算问题的程序设计中计算机操作对象以及操作对象之间的关系和运算操作等的一门学科。它和高级程序设计语言课程不同，高级程序设计语言课程所讨论的最简单程序设计问题，而数据结构课程所讨论的是在数据结构组织的基础上的复杂程序设计问题。

1.1.2 数据结构课程的地位

数据结构作为一门独立的课程在国外是从 1968 年才开始设立的。在 20 世纪 60 年代，美国一些大学的计算机系的教学计划中，只是把数据结构作为一门课程，但对课程的范围没有作明确规定。当时，数据结构几乎是图论特别是表、树的理论的同义词。随后，数据结构这个概念被扩充到包括网络、集合代数论、格、关系等方面，从而变成了现在称之为离散结构的内容。然而，由于数据必须在计算机中进行处理，因此，不仅考虑数据本身的数学性质，而且还必须考虑数据的存储结构，这就进一步扩大了数据结构的内容。

60 年代末到 70 年代初，由于出现了大型程序，软件也相对独立，结构程序设计成为程序设计方法学的主要内容，人们就越来越重视数据结构，认为程序设计的实质是对确定的问题选择一种好的结构，加上设计一种好的算法。1971 年瑞士著名的计算机科学家沃斯(N. Wirth)提出过这样一个著名的公式：算法 + 数据结构 = 程序。从 70 年代中期到 80 年代初，各种版本的数据结构著作相继出现。

目前在我国，数据结构也已经不仅仅是计算机科学技术专业的教学计划中的核心课程之一，而且是其他非计算机专业的主要选修课程之一。

数据结构在计算机科学中是一门综合性的专业基础课。数据结构的研究不仅涉及到计算机硬件(特别是编码理论、存储装置和存储方法等)的研究范围，而且和计算机软件的研究有着密切的关系，无论是编译程序还是操作系统，都涉及到数据元素在存储器中的分配问题。在研究信息检索时也必须考虑如何组织数据，以便查找和存取数据元素更为方便。因此，可以认为数据结构是介于数学、计算机硬件和计算机软件三者之间的一门核心课程。在计算机科学中，数学结构不仅是一般程序设计(特别是非数值计算的程序设计)的基础，而且是设计和实现编译程序、操作系统、数据库系统及其他系统程序和大型应用程序的重要基础。

随着计算机科学的发展，计算机应用的普及，仅仅掌握计算机语言和程序设计的技巧和方法，而缺乏有关数据结构的知识，几乎很难应付复杂的课题，是不能有效利用计算机的。

1.2 基本概念和术语

在计算机科学中，把能被计算机所处理的一切数值的、非数值的信息统称为数据(da-

ta)。例如,数值、字符是数据;声音、图像、图形等也是数据,因为它们经过某些处理后也可被计算机识别存储和处理。

数据元素(data element,简称为元素)是组成数据的基本单位,在有些情况下它又被称为结点、记录等。一个数据元素总是由一个或多个数据项(data item)组成。数据项是数据的不可再分的最小单位,有时它被称为域、字段。如例 1-1 中,职工花名册中的每个职工的信息就是一个数据元素,而每个职工的信息又包括编号、姓名、性别、年龄、月收入等数据项。

数据结构(data structure)指的是数据及数据之间的相互关系。一般来说,数据结构包括以下 3 个方面的内容:

(1) 数据元素之间的逻辑关系,又称为数据的逻辑结构。

(2) 数据元素及其关系在计算机中的表示,又称为数据的物理结构或数据的存储结构。

(3) 数据的运算及实现,即对数据元素可以施加的操作及其这些操作在相应的存储结构上的实现。

数据的逻辑结构是从逻辑关系上描述数据,与数据的存储无关,是独立于计算机的。根据数据元素之间的不同特性,通常有以下 3 类基本逻辑结构:线性结构、树形结构、图结构或网状结构。此外,还有集合这种结构。有时,把树形结构、图结构和集合归纳为非线性结构。

数据的存储结构是指数据元素及其关系存放在计算机内存中的方法。一般来说,数据的存储结构常用的方法有顺序存储方式和链式存储方式,此外,索引存储方式、散列存储方式也较常用。

本书中数据的逻辑结构只介绍线性结构和图、树两种非线性结构;数据的存储结构只介绍顺序存储方式和链式存储方式。

一般来说,一种逻辑结构可采用不同的存储方式。选择何种存储方式要视具体问题而定,也可依据运算是否方便和算法的时间效率和空间要求而定。存储方式是依赖于所采用的计算机语言的,一般可借助高级语言来描述数据的存储结构。

数据的运算是指定义在数据的逻辑结构上的一组操作的集合。例如,一些常用的运算如插入、删除、检索、排序等。至于这些运算的实现在不同的存储方式下是不同的。也就是说运算的实现依赖于所选取的存储结构,依赖于不同的计算机程序设计语言。

1.3 数据类型和抽象数据类型

数据类型是具有相同性质的计算机数据的集合及在这个数据上的这一组运算,是和数据结构密切相关的概念。在用高级程序语言编写的程序中,每个变量、常量或表达式都有一个它所属的确定的数据类型。类型明显或隐含地规定了程序执行期间变量或表达式的所有可能取值的范围,以及在这些值上允许进行的操作。例如,C 语言中的整数类型,它是[min int, max int]区间的整数集合,定义在其上的操作为:加、减、乘、除和取模等算术运算。高级语言中的数据类型可分为两类:一类是非结构的原子类型,它是不可分解的。如:C 语言中的基本类型(整型、字符型和数值型类型),指针类型和空类型。另一类是结构类型。结构类型是由若干成分按某种结构组成的,因此是可以分解的,并且它的成分可以是非结构的,也可以是结构的。例如,C 语言中的结构体,可由许多成员(或称分量)组成。每一个成员称为结构体中的域,可以是整型或字符型,也可以是结构类型。

抽象数据类型(abstract data type,简称为 ADT)是指基于一切逻辑关系的数据类型以及定义在这个类型之上的一组操作。抽象数据类型的定义取决于客观存在的一组逻辑特性,而与其在计算机内的表示和实现的方法无关,即不论其内部结构如何变化,只要它的数学特性不变,都不影响其外部使用。在某种意义上讲,抽象数据类型和数据类型实质上是一个概念。例如,整数类型就是简单的抽象数据类型。因此,"抽象"的意义在于数学特性的抽象。不仅限于各种不同的计算机处理器中已定义并实现的数据类型,还包括设计软件系统时用户自己定义的复杂数据类型。所定义的数据类型的抽象层次越高,含有该抽象数据类型的软件复用程度就越高。

抽象数据类型由元素、结构和操作 3 部分组成。例如,线性表的抽象数据类型可描述如下:

```
ADT Linear_List
{数据元素   a_i 属于同一数据对象, i = 1, 2, …, n (n ≥ 0)
 逻辑结构   a_i 存在次序关系 (a_i, a_{i+1}), a_1 无前驱, a_n 无后继
 操作       设 L 为 Linear_List 类型的线性表,
            InitList(L); /* 建立一个空的线性表 */
            Length(L); /* 求线性表 L 的长度 */
            GetElem(L, i); /* 取线性表 L 中的第 i 个元素 */
            Insert(L, i, x); /* 在线性表 L 中第 i 个元素之前(或之后)插入一个新 */
                             /* 元素 X */
            Delete (L, i); /* 删除线性表 L 中的第 i 个元素 */
}
```

上面介绍的是抽象数据类型线性表的定义。可见,抽象数据类型实际上是数据类型的进一步抽象。即把数据类型和数据类型上的操作(运算)捆在一起,进行封装。而引入抽象数据类型的目的是把数据类型的表示和数据类型上运算的实现与这些数据类型和运算在程序中的引用隔开,使它们相互独立。

1.4 算法描述和算法评价

1.4.1 算法描述

所谓算法(algorithm)是对特定问题求解方法和步骤的一种描述,它是指令和语句的一组有限序列,一个算法必须具备以下 5 个重要特性。

(1) 输入:一个算法必须具有零个或多个的外界输入,这些输入来自于特定对象的集合。

(2) 输出:一个算法必须有一个或多个的输出,这些输出是与输入有着某些特定关系的集合。

(3) 有穷性:一个算法对任何合法的输入值必须总是执行有穷步之后结束,并且每一步都必须在有限时间内完成。

(4) 确定性:算法中每一条指令必须有确切的含义,不会使人在读算法时产生二义性。

(5) 可行性:一个算法是能执行的,即算法中描述的操作都可以通过已经实现的基本运

算执行有限次来实现。

一个算法可以用自然语言、数学语言或约定的符号来描述，也可以用计算机高级语言来描述，如 Pascal 语言、C 语言或类 C 语言等。本书的算法用类 C 语言来描述，这里作如下约定：

(1) 问题的规模用 MAXSIZE 表示，在解决具体问题时可由用户预先定义，例如：

```
#define  MAXSIZE  100;
```

(2) 数据元素的类型约定为 ELEMTP，具体的类型可以由用户在使用时定义，例如：

```
typedef  int  ELEMTP;
```

(3) 数据的存储结构用类型定义(typedef)描述，在书写算法之前进行说明。

(4) 算法以函数形式描述：

```
类型标识符  函数名(形式参数表)
/* 算法说明 */
{语句序列}
```

1.4.2 算法的设计要求

一个好的算法，必须满足以下几方面要求。

(1) 正确性：要经得起一切可能的输入数据的检验；

(2) 可读性：算法要让别人看得懂以便于理解交流；

(3) 健壮性：当输入非法数据时，算法应能作出合理的反应或进行适当的处理；

(4) 高效率：即要求算法执行的时间尽可能短，对于存储空间的需要尽可能少，即省时省空间。

需要说明的是，算法的正确性是必须要满足的，而其他几方面没有一个绝对的判断尺度，必须根据实际问题综合考虑。

1.4.3 算法的评价

解决同一个问题，可以设计出不同的算法，那么，究竟如何来评判一个算法的优劣呢？

显然，由上节的算法设计要求可知，一个好的算法首先应具备正确性，然后是健壮性、可读性。在几个方面都满足的情况下，主要是考虑算法的效率，通过算法的效率高低来衡量不同的算法的优劣程度。

算法的效率指的是执行算法所耗费的时间长短和执行算法所占用空间的多少。

1. 时间

一个算法的执行时间等于其所有语句执行时间的总和，而任一语句的执行时间为该语句的执行次数与该语句执行一次所需时间的乘积。实际情况中每条不同类型语句的执行时间是不同的，但为了问题分析的简单化，一般假定每一条不同类型语句的执行时间相同。因此，讨论一个算法的执行时间等价于讨论所有语句的执行次数。

一般情况下，分析一个算法的执行时间主要依据算法的最大语句频度来计算，它是问题规模 n 的某个函数 $f(n)$，称作算法的时间度量，记作 $T(n) = O(f(n))$，也称为算法的时间复杂度。时间复杂度往往不是精确的执行次数，而是估算的数量级，它着重体现的是随着问题规模 n 的增大，算法执行时间的变化趋势。

例如,在下列3个程序中:

```
(1) x = x + 1;
(2) for (i = 1; i <= n; i++)  x++;
(3) for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)  x++;
```

程序(1)的时间复杂度为O(1);程序(2)的时间复杂度为$O(n)$;程序(3)的时间复杂度为$O(n^2)$。通常分别称O(1)为常量阶,$O(n)$为线性阶,$O(n^2)$为平方阶,此外还可指数阶、对数阶等。显然从时间的角度来看,时间复杂度为常量阶的算法优于线性阶的,线性阶的算法优于平方阶的算法。

2. 空间

空间是指执行算法所需用的存储空间,空间复杂度可类似于时间复杂度的讨论。它与问题规模的函数关系为:$S(n) = O(f(n))$。

思考与练习

1. 简述下列术语:数据、数据元素、数据结构、数据类型。
2. 举例说明逻辑结构、存储结构、运算等方面的内容。
3. 将数量级O(1)、$O(n)$、$O(n^2)$、$O(n^3)$、$O(n\log_2 n)$、$O(\log_2 n)$、$O(2^n)$按增长率由小到大排列。
4. 分析下面程序段的时间复杂度。

```
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    k = i + j;
```

第2章 线性表

在本章及其后续的两章中，将讨论线性结构。线性表、栈、队列和数组的逻辑结构都属于线性结构。线性结构的特点是，数据元素之间的关系是线性的。数据元素可以看成是排列在一条线上或一个环上。在线性结构中，元素之间存在一对一的相互关系，其逻辑特征为：

(1) 存在唯一的一个被称为“开始结点”的数据元素。

(2) 存在唯一的一个被称为“终端结点”的数据元素。

(3) 除开始结点外，其他每个结点有且仅有一个直接前趋(immediate predecessor)。

(4) 除终端结点外，其他每个结点有且仅有一个直接后继(immediate successor)。

本章讨论线性表的逻辑结构、存储结构以及相关的操作。

2.1 线性表的基本概念

2.1.1 线性表的逻辑结构

在实际应用中，线性表(Linear List)是最常用而且最简单的一种数据结构。例如，英文字母(A, B, …, Z)是一个线性表，表中每一个英文字母是一个数据元素。又如表 2-1 所示的学生登记表也是线性表。

表 2-1 学生登记表

学号	专业	姓名	性别	……	年龄
00451101	商业自动化	王 霞	女	……	17
00451203	商业自动化	张文东	男	……	17
00452111	电子商务	陈东高	男	……	18
……	……	……	……	……	……

表中的每个数据元素由若干个数据项组成。综上所述，可以将线性表定义为：线性表是由 n 个 ($n \geqslant 0$) 数据元素组成的有限序列，一般记作：$L = (a_1, a_2, \cdots, a_i, \cdots, a_n)$。表中的元素可以是一个数，一个字母或是由多个数据项组成的复杂信息。同一线性表的数据元素必须具有同一特性，只能属于同一数据对象。

线性表的逻辑结构是通过元素之间的相邻关系体现的：a_1 为开始结点，a_n 为终端结点；a_{i-1} 为 a_i 的直接前趋结点($2 \leqslant i \leqslant n$)；$a_{i+1}$ 为 a_i 的直接后继结点($1 \leqslant i \leqslant n-1$)。线性表中元素的个数($n$) 称为该表的长度。长度为零($n = 0$) 的表称为空表。

2.1.2 线性表的运算

线性表是一种灵活的数据结构，它的长度可根据需要进行增减。对于线性表有以下几种常用的基本运算：

(1) Initlist(L)：建立一个空的线性表 L。

(2) GetElem(L, i)：取线性表 L 中的第 i 个元素。

(3) Length(L)：求线性表 L 的长度。

(4) Locate (L, x)：确定元素 x 在线性表 L 中位置。

(5) Insert(L, i, x)：在线性表 L 中第 i 个元素之前（第 i－1 之后）插入一个新元素 x。

(6) Delete(L, i)：删除线性表 L 中的第 i 个元素。

这里给出的只是定义在逻辑结构上的抽象运算，即只关心这些运算是“做什么”，至于“怎样实现”则依赖于所选定的存储结构。其次，这里给出的只是几种基本的运算，至于其他复杂的运算，如将两个线性表合并成一个线性表，则可以通过这些基本运算的组合来实现。

2.2 线性表的顺序存储

2.2.1 顺序表

如图 2-1 所示，线性表的顺序存储结构指的是用内存中一批地址空间连续的单元依次存储线性表中的数据元素，使得数据元素的逻辑顺序与它在存储器中的物理顺序一致，即让线性表中的第一个元素存放在这个空间的开始位置处，第二个元素紧跟着存放在第一个元素之后，以此类推，这种结构的线性表也称为顺序表。它的特点是线性表中的相邻的元素在内存中的存储位置也是相邻的。由于线性表中的每个元素属于同一类型，所以每个元素在存储器中所占的空间大小相同。在图 2-1 中，假设第一个元素存放的位置为 b，每个元素占用的空间大小为 L，则元素 a_i 的存放位置为：

$$\mathrm{Loc}(a_i) = b + L * (i - 1)$$

一旦起始位置 b 和每个数据元素占用的存储单元的大小 L 确定了下来，就可求出任一

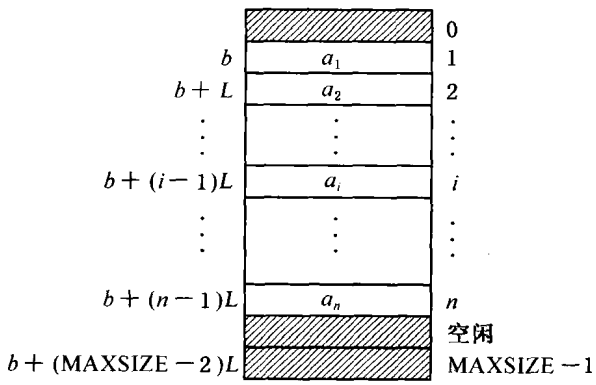

图 2-1 线性表的顺序存储结构

数据元素的存储地址，显然这种表便于进行随机访问。因此，线性表的顺序存储结构是一种随机存取的存储结构。这是顺序存储结构的又一个重要特点。

在高级语言中讨论线性表的顺序存储结构，通常是利用数组来进行描述的。该数组的元素个数，通常是大于线性表的长度，而线性表的长度是经常变化的，用 len 表示当前表长，因此线性表的顺序存储结构可定义为：

```
typedef struct list
{ELEMTP elem[MAXSIZE];   /* 存储线性表中的元素 */
 int len;      /* 线性表的当前表长 */
 }Sqlist;
```

2.2.2 顺序表上的基本操作

在定义线性表的顺序存储结构后，就可以讨论在这种结构上的有关操作问题。在这种存储结构下，某些运算相当容易实现，如 GetElem(L, i)和 Length(L)等。下面重点讨论线性表的数据元素的插入和删除操作。为了讨论方便，不使用数组下标为“0”的单元。

1. 插入

插入是指在线性表中的第 i 个元素之前加入一个新的数据元素 x，使长度为 n 的线性表 $(a_1, a_2, \cdots, a_{i-1}, a_i, \cdots, a_n)$ 变成长度为 $n+1$ 的线性表 $(a_1, a_2, \cdots, a_{i-1}, x, a_i, \cdots, a_n)$。插入时，首先将线性表中原来元素 $a_i, a_{i+1}, \cdots, a_n$ 从 a_n 起依次向后移动一个位置，然后在位置 i 上插入新元素 X。在算法中，还应注意对插入位置的合理性以及表是否已满等特殊情况的处理。

算法 2-1 顺序表上的插入操作。

```
int Insert_sq(Sqlist *L, int i, ELEMTP x)
/* 在线性表的第 i-1 和第 i 元素之间插入一个新元素 */
{ if (i<1 || i>L->len+1)   return 0;          /* 不合理的插入位置 */
  if (L->len == MAXSIZE-1)   return -1;       /* 表已满 */
  for (j = L->len; j>=i; --j)
  L->elem[j+1] = L->elem[j];                  /* 移动 */
  L->elem[i] = x;                             /* 插入 X */
  ++L->len;                                   /* 表长加 1 */
  return 1;
  }  /* Insert_sq */
```

2. 删除

删除是指删除线性表中第 i 个位置上的元素。只需将 $a_{i+1}, \cdots, a_n$ 依次向前移动一个位置，原来长度为 n 的线性表变成长度为 $n-1$ 的线性表。在算法中，还应注意对插入位置的合理性以及表是否已空等特殊情况的处理。

算法 2-2 顺序表上的删除操作。

```
int Delete_sq(Sqlist *L, int i)
/* 删除线性表中第 i 个位置的元素 */
```

```
{ if (i < 1 || i > L->len)   return 0;          /* 不合理的删除位置 */
  if (L->len == 0)   return -1;                 /* 表已空 */
  for (j = i; j <= L->len - 1; j++)
      L->elem[j] = L->elem[j+1];                /* 元素左移 */
  --L->len;                                     /* 表长减 1 */
  return 1;
}  /* Deletet_sq */
```

从插入和删除算法可见，当在顺序表中插入和删除一个元素时，时间主要耗费在元素的移动上，移动元素的次数除了与表长 n 有关外，还与插入和删除的位置有关。

假设 p_i 是在第 i 个元素之前插入一个元素的概率，则在长度为 n 的顺序表中插入一个元素时所需移动元素的平均次数为：

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

其中，当 $1 \leqslant i \leqslant n+1$ 时，$p_1 = p_2 = \cdots = p_{n+1} = 1/(n+1)$

假设 q_i 是删除第 i 个位置的元素的概率，则在长度为 n 的顺序表中删除一个元素时所需移动元素的平均次数为：

$$E_{ds} = \sum_{i=1}^{n} q_i(n-i) = \sum_{i=1}^{n} \frac{1}{n}(n-i) = \frac{n-1}{2}$$

其中，当 $1 \leqslant i \leqslant n$ 时，$q_1 = q_2 = \cdots = q_n = 1/n$

可见，在顺序表中插入和删除一个元素时，大约需要移动表中的一半元素，即算法的时间复杂度为 $O(n)$。

2.2.3 顺序存储结构的特点

由前面的讨论可以看出，采用顺序存储结构的线性表主要有如下特点：

(1) 容易实现，在高级语言中，一般采用一维数组实现。

(2) 内存的存储密度高。

(3) 数据元素的逻辑顺序与它在存储器中的物理顺序一致。

(4) 可以快速随机地存取结点，是一种随机存取的存储结构。

(5) 在插入和删除运算时往往造成大量数据元素的移动，效率较低。

(6) 必须预先为线性表分配空间，表容量难以扩充，必须按线性表最大可能长度分配空间。

2.3 线性表的链式存储结构

本节将讨论线性表的另一种存储结构——链式存储结构，它克服了顺序存储结构的不足，能有效地实现线性表的插入和删除运算，但在下面的讨论中，也会发现在某些方面又不如顺序存储结构。