教育部新世纪网络课程建设工程

# 数据结构

蒋文蓉 编



高等教育出版社
http: // www.hep.com.cn
http: // www.hep.edu.cn

教育部新世纪网络课程建设工程

# 数据结构

蒋文蓉 编

高等教育出版社

## 内容提要

本书是“教育部网络课程建设工程”中《数据结构》网络课程的配套文字教材。

全书十分注重实际应用，针对高职高专教育的特点简化或忽略了一些不常用的数据结构和算法。全书包括绪论、线性表、堆栈和队列、串、多维数组和广义表、树、图、查找、排序等9章。本书既可配套对应的网络课程，也可单独使用。本书还配有演示课件、电子教案、试题库。

本书可作为高等职业院校、高等专科院校、成人高等院校计算机类各专业、信息类及相关专业的教材，也可供非计算机专业学生选用，还可作为本科少学时的参考教材及有关人员自学使用。

**图书在版编目（CIP）数据**

数据结构 / 蒋文蓉编. —北京：高等教育出版社，2003.2
ISBN 7-04-011701-0

Ⅰ. 数… Ⅱ. 蒋… Ⅲ. 数据结构-教材 Ⅳ. TP311.12

中国版本图书馆 CIP 数据核字（2003）第 000426 号

| | | | |
|---|---|---|---|
| 出版发行 | 高等教育出版社 | 购书热线 | 010-64054588 |
| 社　　址 | 北京市东城区沙滩后街55号 | 免费咨询 | 800-810-0598 |
| 邮政编码 | 100009 | 网　　址 | http://www.hep.edu.cn |
| 传　　真 | 010-64014048 | | http://www.hep.com.cn |
| 经　　销 | 新华书店北京发行所 | | |
| 排　　版 | 高等教育出版社照排中心 | | |
| 印　　刷 | 化学工业出版社印刷厂 | | |
| 开　　本 | 787×1092　1/16 | 版　　次 | 2003年2月第1版 |
| 印　　张 | 9 | 印　　次 | 2003年2月第1次印刷 |
| 字　　数 | 210 000 | 定　　价 | 19.00元（含光盘） |

本书如有缺页、倒页、脱页等质量问题，请到所购图书销售部门联系调换。

# 编者的话

本教材是教育部“新世纪网络课程建设工程”中《数据结构》网络课程的配套教材，有很强的针对性，它集理论、实验、习题于一体，是一本较为系统且实用的教材。结合网络课程，本教材既可便于教师教学，又有利于学生学习理解，从而扎实地掌握教学大纲要求的内容。

本教材的最显著特点是有配套的网络课程或单独使用的光盘，因此教学有以下的优点：

1. 趣味性。通过使用优美的音乐、妙趣横生的动画甚至是游戏来贯穿学习过程，激发使用者的学习热情，使读者在学习的同时获得了乐趣。

2. 交互性。可以让使用者直接参与到学习过程当中，并且可以在模拟的操作环境中进行演练。同时，可利用课程提供的网上资源，并通过网络进行交流、讨论。

3. 可控性。学习的进程完全由使用者控制，既可以反复学习某一难点，也可以跳过自己已经理解的部分，或者选择不同的难度，还可以选择学习时间。

4. 实用性。由于师资力量、教学条件、学生程度不同等各方面的制约，很多实际问题和知识都是课堂教学无法较好处理的。但是在丰富的软件资源里，可以满足不同使用者的需要。

5. 灵活性。不受传统地域上的限制，任何人在任何地点都能开始学习，让更多的社会人士不必来到学校就能完成学习。

本教材每一章开始都详细写明了教学要求，方便教师教学和学生学习，使教学更加有针对性，提高了教学的效率，达到事半功倍的效果。而对知识点掌握程度的诠释，如：有的知识点要求学生掌握并且能运用；有的知识点要求学生理解；有的知识点只要求学生知道、了解就可以，使学生在学习的时候分清主次，便于学生抓住学习的重点。

本教材有关算法的部分都配有图形解释或在对应的光盘中配有动画演示，使枯燥的算法变得更加形象和生动，使老师的讲解更容易被学生接受。自学者也可以通过图形解释清楚地了解算法的执行过程。

本教材由上海第二工业大学蒋文蓉编写，本书的习题都配有详细的参考答案（在网上或光盘中），仅供教师参考。在编写过程中，得到了上海第二工业大学陶霖教授的指点和关心，上海第二工业大学软件系的不少教师也给予了热情的帮助，在此谨表示衷心的感谢。

本教材的出版旨在为广大教师和学生提供一块内容全面、重点突出的教学模板，有了这样一本教材，教师可以方便地进行教学，学生也可以比较轻松学习，并从中了解高职高专计算机专业学生应该掌握的有关知识，有的放矢地教与学。

为了及时满足广大读者的需求，进行编写的时间比较仓促，虽然我们尽了很大努力，但难免还会有疏漏和不妥之处，恳请广大教师和读者提出宝贵意见。

编者

2002 年 12 月于上海

# 目　录

# 第一章 绪 论

| | |
|---|---|
| 理论 | 介绍数据结构中常用的基本概念和术语以及学习数据结构的意义 |
| 技能 | |
| 要求 | 了解本章介绍的各种基本概念和术语，掌握算法描述和分析的方法。本章重点是了解数据结构的逻辑结构、存储结构及数据的运算三方面的概念及相互关系，难点是算法复杂度的分析方法 |

## 1.0 教学要求

1. 数据结构的基本概念和术语，要求达到“识记”层次

- 数据、数据元素、数据项、数据结构、数据类型等基本概念
- 数据结构的逻辑结构、存储结构的含义及其相互关系
- 数据结构的两大逻辑结构和四种常用的存储表示方法

2. 数据结构在软件系统中的作用，要求达到“识记”层次

- 数据结构在各种软件系统中所起的作用
- 选择合适的数据结构是解决应用问题的关键步骤

3. 算法的描述和分析，要求达到“领会”层次

- 算法、算法的时间复杂度和空间复杂度、最坏的和平均的时间复杂度等概念
- 算法的时间复杂度不仅仅依赖于问题的规模，也取决于输入实例的初始状态
- 算法描述和算法分析的方法，对于一般算法能分析出时间复杂度

## 1.1 基本概念和术语

这一节先对一些基本概念和术语赋予明确的定义。

**数据（Data）**：信息的载体，它能够被计算机识别、存储和加工处理。对计算机科学而言，数据的含义极为广泛，比如数字、文字、图形、图像、色彩、声音都可以通过编码而归于数据的范畴。

**数据元素（Data Element）**：是数据的基本单位。数据元素有时也称为结点、记录等。一个数据元素一般由若干个数据项组成。

**数据项（Data Item）**：具有独立意义的最小数据单位，是对数据元素属性的描述。数据项也称域或字段。例如：在学生档案管理系统中，可以把一个学生的有关信息作为一个数据元素，它由学号、姓名、年龄等数据项

组成。

**数据结构**（**Data Structure**）：指的是数据之间的相互关系，即数据的组织形式。

**数据的逻辑结构**（**Logical Structrue of Data**）：数据元素之间的逻辑关系。

数据的逻辑结构可分为两大类：线性结构和非线性结构。

（1）线性结构：特点是结构中有且仅有一个始结点和一个终结点，始结点只有一个后继结点，终结点只有一个前趋结点，每个内结点有且仅有一个前趋结点和一个后继结点。线性结构最一般的情形是线性表（见图 1－1）。

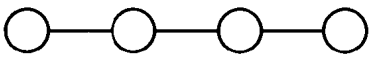


图 1－1　线性结构

（2）非线性结构：特点是结构中的结点可能有多个前趋结点和多个后继结点（见图 1－2）。

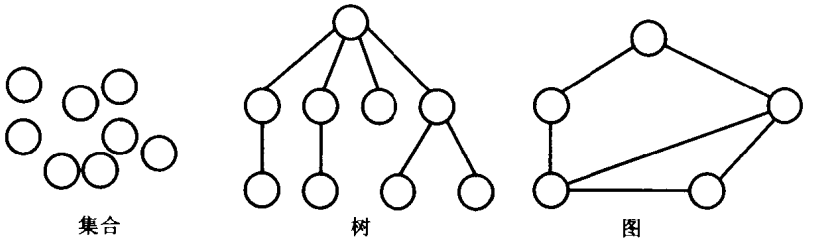



图 1－2　非线性结构

1）集合：结构中的数据元素之间除了“同属于一个集合”外，别无其他的关系。

2）树：结构中的数据元素之间存在一个对多个的关系。

3）图(网)：结构中的数据元素之间存在多个对多个的关系。

**数据的存储结构**（**Storage Structure**）：数据元素及其关系在计算机存储器内的表示。

数据的存储结构可用以下四种基本的存储方法得到：顺序存储方法、链接存储方法、索引存储方法和散列存储方法。数据逻辑结构和存储结构是数据结构的两个密切相关的方面，同一逻辑结构可以对应不同的存储结构。算法的设计取决于设计的逻辑结构，而算法的实现依赖于指定的存储结构。

**数据的运算**（**Data Calculate**）：即对数据施加的操作。

**数据类型**（**Data Type**）：是一个值的集合以及在这些值上定义的一组操作的总称。

各种高级语言提供的基本数据类型有所不同，可以分为两类：

**(1) 原子类型**（**Atom Type**）：其值不可分解，如 C 语言的整型、字符型等标准类型及指针等简单的导出类型；通常是由语言直接提供的。

**(2) 结构类型**（**Structure Type**）：其值可分解为若干个成分（或称为分量），如 C 语言的数组、结构等类型；通常是由标准类型派生的，故它也是

一种导出类型。

**抽象数据类型（Abstract Data Type）**：简称 ADT，是指抽象数据的组织和与之相关的操作。

```
ADT 抽象数据类型名
{
  数据对象：〈数据对象的定义〉
  数据关系：〈数据关系的定义〉
  基本操作：〈基本操作的定义〉
}
```

## 1.2 学习数据结构的意义

尼克劳斯·沃思（Niklaus Wirth）瑞士计算机科学家，PASCAL 之父及结构化程序设计的首创者，获得 1987 年计算机先驱奖。

数据结构是一门介于数学、计算机硬件和计算机软件三者之间的计算机专业核心课程。

在计算机科学中，数据结构不仅是一般程序设计的基础，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。

学习“数据结构”既为进一步学习其他计算机专业课程提供必要的准备知识，也有助于提高软件设计和程序编制水平。

瑞士著名的计算机科学家沃思（N.Wirth）教授曾提出：**算法 + 数据结构 = 程序**。这里的数据结构是指数据的逻辑结构和存储结构，而算法则是对数据运算的描述。由此可见，程序设计的实质是针对实际问题的要求，合理地选择数据结构和算法，而好的算法在很大程度上取决于描述实际问题的数据结构。请看下面的几个例子。

**【例 1】** 人机对弈问题

计算机能够和人进行对弈，是因为有人将对弈的策略存入计算机。而对弈是在一定规则下随机进行的，不仅要看棋盘当时的格局，还要能够预测将来可能发生的情况。这是一个相当复杂的问题，不仅需要数据结构的知识，还深入涉及到人工智能的领域。我们在这里举一个简单例子，只是想说明数据结构的广泛用途。具体对弈过程参阅“配套光盘”。

**人机对弈问题**

请您用鼠标点您所希望落子的方格，您所下的棋子用“●”表示，计算机选手所下的棋子用“○”表示。

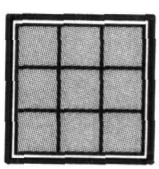


图 1-3 人机对弈问题

【例 2】 Hanoi 塔问题

假设有三个分别命名为 A、B 和 C 的塔座，在塔座 B 上插有 $n$ 个直径大小各不相同、依小到大编号为 1,2,…,$n$ 的圆盘。现要求将 B 轴上的 $n$ 个圆盘移至塔座 A 上并仍按同样顺序叠排，圆盘移动时必须遵循下列规则：

(1) 每次只能移动一个圆盘；

(2) 圆盘可以插在 A、B 和 C 中的任一塔座上；

(3) 任何时刻都不能将一个较大的圆盘压在较小的圆盘之上。

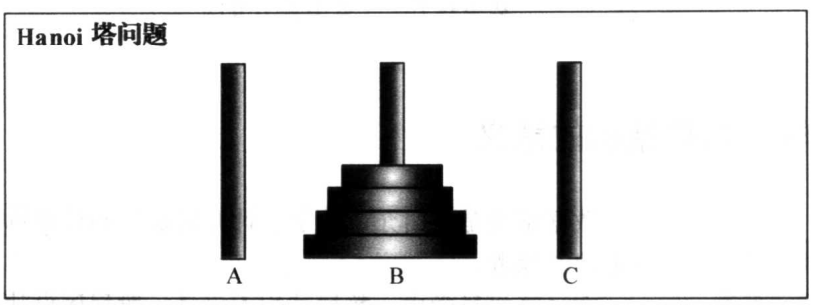



图 1-4 Hanoi 塔问题

## 1.3 算法的描述和分析

**算法**（**Algorithm**）：是对特定问题求解采取的方法和步骤，也就是说，算法是为实现某一计算过程而制定的基本动作的执行序列。它是指令的有限序列，其中每一条指令表示一个或多个操作。一个算法一般具有下列五个重要特性：

(1) 有穷性：一个算法必须总是在执行有限步之后结束。

(2) 确定性：算法中的每一条指令必须有确切的含义，不能产生多义性。

(3) 可行性：算法中的每一条指令必须是切实可行的，即原则上是可以通过已经实现的基本运算执行有限次来实现的。

(4) 输入：一个算法有零个或多个输入，这些输入取自于特定对象的集合。

(5) 输出：一个算法有一个或多个输出，这些输出是同输入有某个特定关系的量。

描述算法的一些要求和规则：

(1) **正确性**：“正确”一词在含义上大体分为四个层次：

1) 程序不含语法错误；

2) 程序对几组输入数据能给出满足规格说明要求的结果；

3) 程序对精心选择的、典型的、苛刻而带有刁难性的几组输入数据能给出满足规格说明要求的结果；

4）程序对一切合法的输入数据都能给出满足规格说明要求的结果。

(2) **可读性**：算法主要是为了人的阅读，其次才是机器的执行。可读性有助于人对算法的理解。

(3) **健壮性**：当输入数据非法时，算法也能适当的作出反应或进行处理，而不会简单地拒绝或任许错误执行。

(4) **效率和低存储量需求**：一般来说，效率指的是算法执行的时间；存储量指的是算法执行过程中需要的最大存储空间。这两者都与输入的规模和输入数据的性质有关。

其中“正确性”要求是其他所有要求的基础，离开了正确性，其他一切问题都无从谈起。其次“算法的效率”是一切算法设计者所追寻的目标：算法的效率直接影响算法在实际情况中的执行，效率的高低很可能就决定了这个算法是否有实用价值。

一个算法可以使用自然语言、数学语言或约定的符号语言来描述，本书采用C语言作为算法描述语言。

一种数据结构的优劣是由实现其各种运算的算法体现的。评价一个算法主要看这个算法所要占用的机器资源的多少。而在这些资源中，时间和空间是两个最主要的方面，因此算法分析中最关心的也就是算法所需要的时间消耗和空间占用量，即算法的时间复杂度和空间复杂度。

**算法的空间复杂度**（**Space Complexity**）指的是：当问题的规模以某种单位由1增至 $n$ 时，解决该问题的算法实现所占用的空间也以某种单位由1增至 $f(n)$，则称该算法的空间复杂度是 $f(n)$。

**语句频度**（**Frequency Count**）指的是该语句重复执行的次数。

**算法的时间复杂度**（**Time Complexity**）指的是：算法中基本操作重复执行的次数依据算法中最大语句频度来估算，它是问题规模 $n$ 的某个函数 $f(n)$，算法的时间量度记作 $T(n)=O(f(n))$，表示随问题规模 $n$ 的增大，算法执行时间的增长度和 $f(n)$ 的增长度相同。

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。

常用时间复杂度有如下关系：

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

## 思考题

猴子吃桃子问题。猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个；第二天早上又将剩下的桃子吃掉了一半，又多吃了一个；以后每天早上都吃了前一天剩下的一半加一个，到第十天早上想再吃时，发现只剩下一个桃子了。求第一天共摘了多少个桃子。

# 第二章　线　性　表

| | |
|---|---|
| 理论 | 介绍线性表的逻辑结构和各种存储表示方法，以及定义在逻辑结构上的各种基本运算及其在存储结构上如何实现这些基本运算 |
| 技能 | 顺序表的操作<br>链表的操作 |
| 要求 | 在熟悉这些内容的基础上，能够针对具体应用问题的要求和性质，选择合适的存储结构，设计出相应的有效算法，解决与线性表相关的实际问题。本章重点是熟练掌握顺序表和单链表上实现的各种基本算法及相关的时间性能分析，难点是能够使用本章所学到的基本知识设计有效算法，解决与线性表相关的应用问题 |

## 2.0 教学要求

1. 线性表的逻辑结构，要求达到“识记”层次

- 线性表的逻辑结构特征
- 线性表上定义的基本运算，并能利用基本运算构造出较复杂的运算

2. 线性表的顺序存储结构，要求达到“综合应用”层次

- 顺序表的含义及特点，即顺序表如何反映线性表中元素之间的逻辑关系
- 顺序表上的插入、删除操作及其平均时间性能分析
- 利用顺序表设计算法，解决简单的应用问题

3. 线性表的链式存储结构，要求达到“综合应用”层次

- 链表如何表示线性表中元素之间的逻辑关系
- 链表中头指针和头结点的使用
- 单链表、双（向）链表、循环链表链接方式上的区别
- 单链表上实现的建表、查找、插入和删除等基本算法，并分析其时间复杂度
- 循环链表上尾指针取代头指针的作用，以及单循环链表上的算法与单链表上相应算法的异同点
- 双链表的定义及其相关算法
- 利用链表设计算法，解决简单的应用问题

4. 顺序表和链表的比较，要求达到“领会”层次

- 顺序表和链表的主要优缺点
- 针对线性表上所需要执行的主要操作，知道选择顺序表还是链表作为其存储结构才能取得较优的时空性能

5. 实训

- 顺序表的操作

  顺序表的查找、插入与删除。设计算法，实现线性结构上的顺序表的建立以及元素的查找、插入与删除。

- 链表的操作

  单链表的查找、插入与删除。设计算法，实现线性结构上的单链表的建立以及元素的查找、插入与删除。

## 2.1 线性表的逻辑结构

**线性表（Linear List）**是由 $n$（$n>0$）个性质相同的数据元素组成的有限序列。一般记为：

$$(a_1,\ a_2,\ a_3,\ \ldots,\ a_n)$$

表中数据元素的个数 $n$ 定义为线性表的长度。$n=0$ 的表称为空表，即该线性表不包含任何数据元素。

线性表中的数据元素可以是简单类型，也可以是由用户定义的任何类型，如整型、实型、字符型、记录类型等。

线性表是一种最简单、最常用的数据结构。

线性结构的特点：在数据元素的非空有限集合中，①存在惟一的一个被称做“第一个”的数据元素；②存在惟一的一个被称做“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，集合中每个数据元素均只有一个后继。

**【例 1】**（1，2，3，4，5，6，7，8）是一个线性表，其中的数据元素是数字，共有 8 个数据元素。其中“1”是第一个数据元素，“8”是最后一个数据元素，“1”是“2”的直接前驱，“2”是“1”的直接后继。“1”没有直接前驱，“8”没有直接后继。

**【例 2】** 表 2-1 所示的学生成绩表也是一个线性表，其中的数据元素是每个学生所对应的一行信息，它是由学号、姓名、期中考试成绩、期末考试成绩和总成绩共 5 个数据项组成。通常把这种数据元素称为记录。表中所列出的学生人数 30 就是该表中数据元素的个数。

**表 2-1 学生成绩表**

| 学号 | 姓名 | 期中考试成绩 | 期末考试成绩 | 总成绩 |
|---|---|---|---|---|
| 1 | 王五 | 81 | 89 | 85 |
| 2 | 张三 | 66 | 96 | 81 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 30 | 李四 | 70 | 56 | 63 |

线性表的两类存储结构：顺序存储结构(顺序表)和链式存储结构(链表)。

常见的线性表的基本运算有如下六种：

(1) InitList (L)

构造一个空的线性表 L，即表的初始化。

(2) ListLength (L)

求线性表 L 中的结点个数，即求表长。

(3) GetNode (L, i)

取线性表 L 中的第 $i$ 个结点，这里要求 $1 \leqslant i \leqslant$ ListLength (L)。

(4) LocateNode (L, x)

在 L 中查找值为 $x$ 的结点，并返回该结点在 L 中的位置。若 L 中有多个结点的值和 $x$ 相同，则返回首次找到的结点位置；若 L 中没有结点的值为 $x$，则返回一个特殊值表示查找失败。

(5) InsertList (L, x, i)

在线性表 L 的第 $i$ 个位置上插入一个值为 $x$ 的新结点，使得原编号为 $i$，$i+1$，…，$n$ 的结点变为编号为 $i+1$，$i+2$，…，$n+1$ 的结点。这里 $1 \leqslant i \leqslant n+1$，而 $n$ 是原表 L 的长度。插入后表 L 的长度加 1。

(6) DeleteList (L, i)

删除线性表 L 的第 $i$ 个结点，使得原编号为 $i+1$，$i+2$，…，$n$ 的结点变成编号为 $i$，$i+1$，…，$n-1$ 的结点。这里 $1 \leqslant i \leqslant n$，而 $n$ 是原表 L 的长度。删除后表 L 的长度减 1。

## 2.2 线性表的顺序存储结构

### 2.2.1 顺序表

**顺序表 (Sequential List)**：线性表顺序存储结构的简称，是计算机中最简单和最常用的一种存储方式，即用一组连续的存储单元依次存放线性表的数据元素。

若每个数据元素占用 $c$ 个存储单元，并以所占的第一个存储单元地址作为

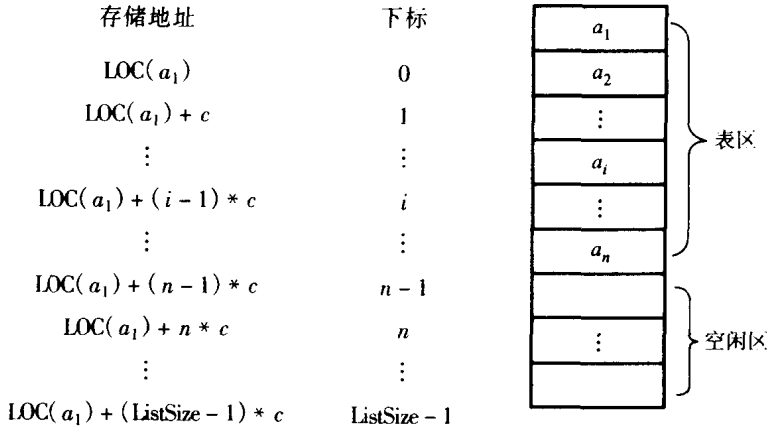



图 2-1 线性表顺序存储结构示意图

这个数据元素的存储位置（见图2-1所示），则表中任一元素 $a_i$ 的存储地址为：

$$LOC(a_i) = LOC(a_1) + (i-1) * c \qquad 1 \leqslant i \leqslant n$$

顺序表的特点是表中相邻的元素 $a_i$ 和 $a_{i+1}$ 赋以相邻的存储位置 $LOC(a_i)$ 和 $LOC(a_{i+1})$。即在线性表中逻辑关系相邻的数据元素，在计算机内存中的物理位置也是相邻的。对于这种存储方式，只要确定了存储线性表的起始位置，线性表中任一数据元素都可随机存取，所以顺序表是一种随机存取的存储结构。

### 2.2.2 顺序表上实现的基本运算

1. 建立

由于程序语言中的向量（一组数组）是采用顺序存储表示，故可用向量这种数组类型来描述顺序表。我们用结构类型来定义顺序表类型，如下：

```
//顺序表的定义
# define ListSize 100
  //表空间大小可根据实际需要而定，这里假设为 100
typedef int DataType;
  //DataType 可以是任何相应的数据类型如 int，float 或 char
typedef struct
{
  DataType data[ListSize];//向量 data 用于存放表结点
  int length;              //当前的表长度
}SeqList;
```

输入 $n$ 个整数，产生一个存储这些整数的顺序表 L 的函数，如下：

**算法2-1 顺序表的建立**

```
void CreateList (SeqList * L, int n)
{
  int i;
  for (i=0; i<n; i++)
    scanf ("% d", &L->data [i]);
  L->length = n;
}
```

2. 查找

在一个顺序表中查找元素值为 $x$ 的元素的函数，如下：

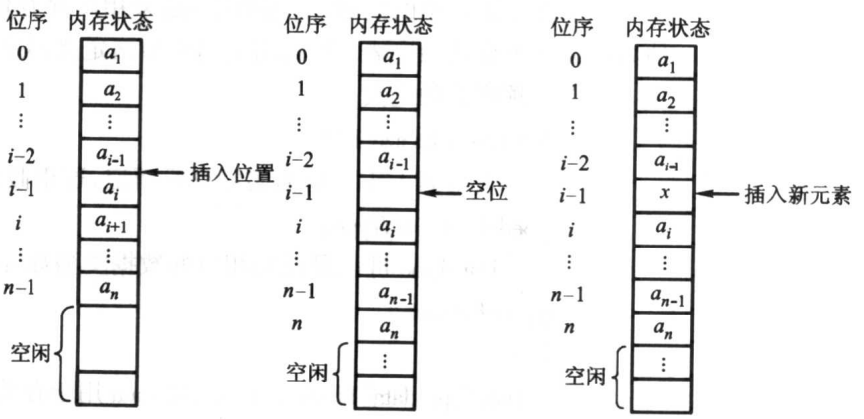
**算法2-2 顺序表的查找**

```
int LocateList (SeqList L, DataType x)
{
  int i=0;
```

```
  while (i < L. length && L. data[i] ! = x)
    ++i;
  if (i < L. length)
    return i + 1;        //找到返回该值的位置
  else return 0;         //找不到返回 0
}
```

3. 插入

线性表的插入运算是指在表的第 $i$（$1 \leqslant i \leqslant n$ 个位置上，插入一个新结点 $x$，使长度为 $n$ 的线性表（$a_1$，…，$a_{i-1}$，$a_i$，…，$a_n$）变成长度为 $n+1$ 的线性表（$a_1$，…，$a_{i-1}$，$a_x$，$a_i$，…，$a_n$）（见图 2-2）。




(a) 输入新元素前　　(b) 插入前，移出空位以后　　(c) 插入新元素 $x$ 后

图 2-2　顺序表插入新元素的过程示意图

插入操作分成两阶段：第一阶段将位于插入点以后的数据元素依次向后移动，为新数据元素腾出一个空位，然后在第二阶段中将数据元素插入空位。

在一个顺序表中第 $i$ 个元素之前插入一个元素 $x$ 的函数，如下：

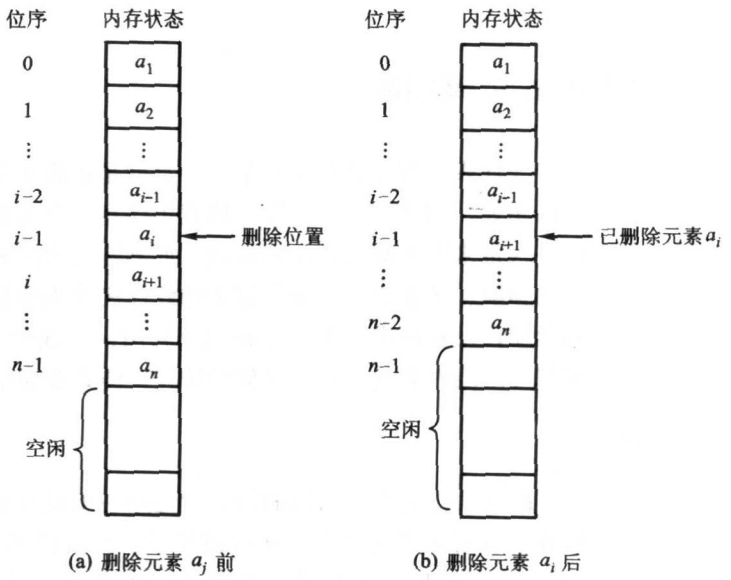
**算法 2-3　顺序表的插入**

```
void InsertList (seqList * L, DataType x, int i)
{
  //将新结点 x 插入 L 所指的顺序表的第 i 个结点的位置上
  int j;
  if (i < 1 || i > L -> length + 1)
  {
    printf("插入位置非法 \ n");
    exit(0);
  }
  if (L -> length > = ListSize)
```

```
{
  printf("表空间溢出,退出运行\n");
  exit(0);
}
  for (j = L->length - 1; j>=i-1;j--)
    L->data[j+1] = L->data[j];//从表尾元素到第 i 个元素逐个后移
  L->data[i-1] = x;                    //新元素插入
  L->length++;                         //修正表长
}
```

4. 删除

线性表的删除运算是指将表的第 $i$（$1 \leqslant i \leqslant n$）个位置上的结点删除，使长度为 $n$ 的线性表（$a_1$，…，$a_{i-1}$，$a_i$，…，$a_n$）变成长度为 $n-1$ 的线性表（$a_1$，…，$a_{i-1}$，$a_{i+1}$，…，$a_n$）（见图 2-3）。




(a) 删除元素 $a_j$ 前　　(b) 删除元素 $a_i$ 后

图 2-3　顺序表删除一个元素的过程示意图

删除操作分为相应两个阶段，只是顺序与插入运算相反：第一阶段先执行数据元素的删除，第二阶段再依次向前移动数据将空位填上。

在一个顺序表中删除第 $i$ 个元素的函数，如下：

**算法 2-4　顺序表的删除**

```
void DeleteList(SeqList *L,int i)
{
  //从 L 所指的顺序表中删除第 i 个结点
  int j;
```