

高等学校计算机应用规划教材

数 据 结 构

(C 语言版)

梁海英 王凤领 主 编
谭晓东 巫湘林 张 波 胡元闯 副主编

清华大学出版社

北 京

内 容 简 介

本书基于我们多年的教学经验,从实用的角度出发,对线性和非线性数据结构的顺序和链式存储及其操作进行了详细讲解。书中的每一章均配有实践练习及大量习题,实现了理论与实践相结合,让学生学以致用。本书免费提供电子课件、源程序及习题答案,全部案例均在 Visual C++ 6.0 环境中成功运行。

本书既可作为普通高校非计算机专业的计算机公共课教材、计算机类应用型本科及专科教材,也可作为计算机软件考试的优秀教材。

本书对应的电子课件、习题答案和源代码可以到 <http://www.tupwk.com.cn> 网站下载。

本书封面贴有清华大学出版社防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话:010-62782989 13701121933

图书在版编目(CIP)数据

数据结构: C 语言版 / 梁海英, 王凤领 主编. —北京: 清华大学出版社, 2017

(高等学校计算机应用规划教材)

ISBN 978-7-302-47979-6

I. ①数… II. ①梁… ②王… III. ①数据结构—高等学校—教材 ②C语言—程序设计—高等学校—教材 IV. ①TP311.12 ②TP312.8

中国版本图书馆 CIP 数据核字(2017)第 201113 号

责任编辑: 胡辰浩 马玉萍

装帧设计: 孔祥峰

责任校对: 曹 阳

责任印制:

出版发行: 清华大学出版社

网 址: <http://www.tup.com.cn>, <http://www.wqbook.com>

地 址: 北京清华大学学研大厦 A 座 邮 编: 100084

社 总 机: 010-62770175 邮 购: 010-62786544

投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn

质 量 反 馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn

印 刷 者:

装 订 者:

经 销: 全国新华书店

开 本: 185mm×260mm 印 张: 21 字 数: 511 千字

版 次: 2017 年 8 月第 1 版 印 次: 2017 年 8 月第 1 次印刷

印 数: 1~2500

定 价: 48.00 元

产品编号:

前 言

数据结构在计算机科学中是一门综合性的专业基础课，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。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的重要基础。目前比较权威的数据结构教材大多是考研指定教材，难度比较大，不太适合应用型本科、三本及专科学生使用。为此，我们编写了这本教材，通过引入大量案例，将复杂的理论问题直观化，案例驱动式教学，更有利于这个层次的学生接受。

我们基于多年的丰富的教学经验及素材积累，精心编写此书，目的是让初学者能循序渐进地掌握各种数据结构及操作，力求透彻、全面、易学、易用，充分调动学生的学习积极性。书中使用 C 语言定义各种数据结构、描述算法。本书对每种数据结构和算法的剖析都遵循由浅入深的原则，并配以实用的案例和图示，配有相应的 C 语言源代码，适合具有 C 语言基础的数据结构初学者。

全书共分 8 章，对于常用的数据结构，如线性表、栈、队列、串、矩阵、广义表、树、二叉树、图等进行深入讲解，使读者能够全面地理解基本概念、逻辑结构、存储结构、操作运算、实现算法以及案例应用，进而利用比较法讲解各种查找和排序的方法，并对各种算法的性能进行分析，以便在不同的应用场合选取合适的方法。

本书由梁海英博士和王凤领教授任主编，谭晓东、巫湘林、张波和胡元闯任副主编，全书由贺州学院梁海英教授统稿。在本书编写过程中，得到了所在学院的同事的热心帮助和支持，参加本书内容编写、程序调试、课件制作、习题收集、答案制作、内容审校等工作的老师有赵方珍、罗兰花、李立信、千文、黄华升、陈冠萍、袁淑丹等，在此向他们表示衷心的感谢！

由于水平有限，书中难免存在不妥之处，敬请读者谅解，并提出宝贵意见。我们的电话是 010-62796045，信箱是 huchenhao@263.net。

本书对应的电子课件、习题答案和源代码可以到 <http://www.tupwk.com.cn> 网站下载。

编 者

2017 年 5 月

目 录

第 1 章 绪论	1	2.5 本章小结	53
1.1 数据结构概述	1	2.6 习题 2	54
1.2 常用术语和基本概念	3	第 3 章 栈和队列	59
1.3 数据类型	6	3.1 栈	59
1.3.1 数据类型概述	6	3.1.1 栈的定义	59
1.3.2 抽象数据类型	7	3.1.2 栈的顺序存储与操作	61
1.4 算法和算法复杂度	8	3.1.3 栈的链式存储与操作	65
1.4.1 算法的重要性	9	3.2 队列	66
1.4.2 时间复杂度	10	3.2.1 队列的定义	67
1.4.3 空间复杂度	12	3.2.2 队列的顺序存储与操作	68
1.5 本章实战练习	13	3.2.3 队列的链式存储与操作	71
1.6 本章小结	15	3.3 本章实战练习	73
1.7 习题 1	16	3.3.1 top 为指针且指向栈顶 元素的下一个位置	73
第 2 章 线性表	19	3.3.2 top 为整数且指向栈顶 元素的下一个位置	75
2.1 线性表概述	19	3.3.3 栈的应用——数制转换	77
2.1.1 线性表的定义及特点	19	3.3.4 顺序队列的基本操作	79
2.1.2 线性表的抽象数据类型的 定义	20	3.3.5 循环队列设置不同队空与 队满条件的解决方案	82
2.2 线性表的顺序存储及运算的 实现	21	3.3.6 链队列的基本操作	85
2.2.1 线性表的顺序存储	21	3.4 本章小结	88
2.2.2 顺序表的基本操作	22	3.5 习题 3	89
2.3 线性表的链式存储及运算的 实现	28	第 4 章 串、数组、矩阵和广义表	94
2.3.1 单链表	28	4.1 串的定义	95
2.3.2 循环链表	37	4.1.1 串的基本概念	95
2.3.3 双向链表	38	4.1.2 串的抽象数据类型的定义	96
2.4 本章实战练习	41	4.2 串的存储与操作	97
2.4.1 顺序表的常用操作	41	4.2.1 串的顺序存储与操作	97
2.4.2 单链表的常用操作	45	4.2.2 串的链式存储与操作	99
2.4.3 通讯录管理	47	4.3 数组	100

4.3.1 数组的定义·····	100	5.6 哈夫曼树·····	159
4.3.2 数组的顺序存储·····	101	5.6.1 哈夫曼树的定义·····	159
4.4 矩阵的压缩存储·····	102	5.6.2 哈夫曼树的构造算法·····	159
4.4.1 特殊矩阵的压缩存储·····	102	5.6.3 哈夫曼编码·····	161
4.4.2 稀疏矩阵及其压缩存储·····	105	5.7 本章实战练习·····	162
4.5 广义表·····	111	5.7.1 二叉树的基本操作·····	162
4.5.1 广义表的定义·····	111	5.7.2 线索二叉树的操作·····	167
4.5.2 广义表的存储结构及 实现·····	112	5.7.3 树的应用——模拟资源 管理器·····	171
4.6 本章实战练习·····	114	5.7.4 哈夫曼树构造·····	178
4.6.1 串的常见操作·····	114	5.8 本章小结·····	180
4.6.2 串的基本操作及应用·····	117	5.9 习题5·····	180
4.6.3 数组应用——方阵·····	126	第6章 图·····	189
4.6.4 数组应用——稀疏矩阵·····	126	6.1 图的定义和基本术语·····	189
4.7 本章小结·····	130	6.1.1 图的定义·····	189
4.8 习题4·····	131	6.1.2 图的基本术语·····	190
第5章 树·····	134	6.2 图的存储与操作·····	194
5.1 树的概念·····	134	6.2.1 邻接矩阵·····	194
5.1.1 树的定义·····	134	6.2.2 邻接表·····	197
5.1.2 树的基本术语·····	135	6.2.3 十字链表·····	200
5.2 二叉树·····	137	6.3 图的遍历·····	201
5.2.1 二叉树的定义·····	137	6.3.1 深度优先遍历算法·····	202
5.2.2 二叉树的性质·····	139	6.3.2 广度优先遍历算法·····	204
5.3 二叉树的存储结构·····	141	6.4 图与最小生成树·····	206
5.3.1 二叉树的顺序存储·····	141	6.4.1 生成树和森林的算法·····	206
5.3.2 二叉树的链式存储与 操作·····	142	6.4.2 最小生成树·····	208
5.4 二叉树的遍历·····	145	6.5 最短路径·····	212
5.4.1 遍历算法·····	145	6.5.1 单源点到其余各顶点的 最短路径·····	212
5.4.2 线索二叉树·····	148	6.5.2 任意源点之间的最短 路径·····	217
5.4.3 遍历算法的应用举例·····	152	6.6 AOV网与拓扑排序·····	218
5.5 树与森林·····	153	6.6.1 AOV网·····	219
5.5.1 树和森林的存储·····	154	6.6.2 拓扑排序·····	220
5.5.2 二叉树、树和森林的 转换·····	157	6.7 AOE网与关键路径·····	222
5.5.3 树和森林的遍历·····	158	6.7.1 AOE网·····	222

6.7.2 关键路径·····	223	7.4.4 哈希表的查找和性能·····	285
6.8 本章实战练习·····	226	7.5 本章实战练习·····	286
6.8.1 图的邻接矩阵操作·····	226	7.5.1 顺序查找算法·····	286
6.8.2 图的邻接表操作·····	230	7.5.2 折半查找算法·····	287
6.8.3 利用邻接矩阵实现连通图 的深度优先遍历·····	234	7.5.3 二叉排序树查找算法·····	289
6.8.4 利用邻接表实现连通图 的深度优先遍历·····	237	7.6 本章小结·····	291
6.8.5 利用邻接矩阵实现连通图 的广度优先遍历·····	239	7.7 习题 7·····	292
6.8.6 利用邻接表实现连通图 的广度优先遍历·····	242	第 8 章 排序·····	296
6.8.7 普里姆最小生成树算法·····	245	8.1 排序的定义·····	296
6.8.8 迪杰斯特拉最短路径 算法·····	248	8.2 插入排序算法·····	297
6.9 本章小结·····	251	8.2.1 直接插入排序·····	298
6.10 习题 6·····	251	8.2.2 折半插入排序·····	299
第 7 章 查找·····	260	8.2.3 二路插入排序·····	301
7.1 查找的定义·····	260	8.2.4 表插入排序·····	303
7.2 静态查找算法·····	262	8.2.5 希尔排序·····	304
7.2.1 顺序查找·····	262	8.3 交换排序算法·····	305
7.2.2 折半查找·····	265	8.3.1 冒泡排序·····	305
7.2.3 分块查找·····	268	8.3.2 快速排序·····	306
7.3 动态查找算法·····	270	8.4 选择排序算法·····	309
7.3.1 二叉排序树·····	270	8.4.1 直接选择排序·····	309
7.3.2 平衡二叉树·····	275	8.4.2 堆排序·····	310
7.4 哈希表·····	279	8.5 归并排序算法·····	314
7.4.1 哈希表的定义·····	279	8.6 排序算法的比较·····	316
7.4.2 哈希函数的构造·····	280	8.7 本章实战练习·····	317
7.4.3 处理冲突的方法·····	283	8.8 本章小结·····	322
		8.9 习题 8·····	323
		参考文献·····	327

第1章 绪 论

本章对数据结构进行简要概述，主要介绍数据结构的相关概念以及它所研究的问题与内容，目的是让读者对数据结构有个大致的了解，为后续内容的学习提供必要的基础知识。本章介绍的主要内容包括：数据结构概述、常用术语和基本概念、数据类型、算法和算法复杂度。

1. 总体要求

了解数据结构的意义，数据结构在计算机领域中的地位和作用；掌握数据结构各名词、术语的含义和有关的基本概念；数据的逻辑结构和存储结构之间的关系；了解使用 C 语言对数据结构进行抽象数据类型的存储和实现的方法；了解算法的五个特征；掌握通过计算语句执行次数来估算算法时间复杂度的方法。

2. 相关知识点

相关术语：数据、数据元素、数据项、数据对象、数据结构。数据的逻辑结构：集合、线性结构、树和图。数据的物理结构：顺序和非顺序结构。算法的五个特征、时间复杂度及空间复杂度。

3. 学习重点

数据的逻辑结构和存储结构及其之间的关系；算法时间复杂度、空间复杂度及其计算。

1.1 数据结构概述

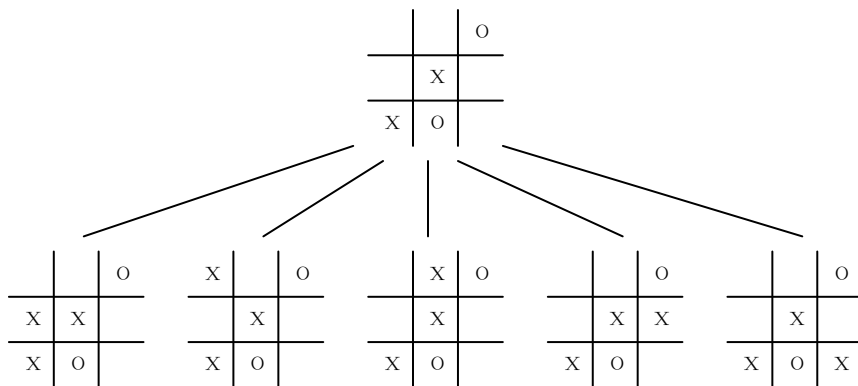
“数据结构(Data Structure)+算法(Algorithm)=程序(Program)。”计算机算法与数据的结构密切相关，算法无不依附于具体的数据结构，数据结构直接关系到算法的选择和效率。运算由计算机来完成，这就需要设计相应的插入、删除和修改的算法。也就是说，数据结构还需要给出每种结构类型所定义的各种运算的算法。

一般来说，用计算机来解决一个具体问题时，大致需要经过以下几个步骤：首先，要从具体问题中抽象出一个适当的数学模型，然后设计一个解此数学模型的算法(Algorithm)，最后编写程序、进行测试，调试和运行直到得到最终解答。

寻求数学模型的实质是分析问题，从中提取操作的对象，并找出这些操作对象之间含有的关系，然后用数学的语言加以描述。当人们用计算机处理数值计算问题时，所用的数学模型是用数学方程描述的。所涉及的运算对象一般是简单的整型、实型和逻辑型数据，因此程序设计者的主要精力集中于程序设计技巧上，而不是数据的存储和组织上。然而，

		O
	X	
X		O

(a) 棋盘格局示例



(b) 对弈树的局部

图 1.2 人机对弈

1.2 常用术语和基本概念

要学好“数据结构”这门课程，必须要明确各种概念及其相互之间的关系。本节只介绍一些常用的术语和基本概念，其他的相关术语和概念将在以后各章节中陆续讲解。

1. 数据

数据(Data)是信息的载体，是可以被计算机识别、存储并加工处理的描述客观事物的信息符号的总称。在计算机学科中，数据是指所有能输入到计算机中，并能被计算机程序所处理的符号的集合，它是计算机程序加工处理的对象。因此，客观事物包括数值、字符、声音、图形、图像等，它们本身并不是数据，只有通过编码变成能被计算机识别、存储和处理的符号后才是数据。

2. 数据元素和数据项

数据元素(Data Element)是描述数据的基本单位，一个数据元素由若干个数据项组成。数据项(Data Item)是描述数据的最小单位。在计算机中存储数据时，都是以一个数据元素为单位的。

数据项分为两种类型：组合项和原子项。

(1) 组合项

若数据元素可再分，则每一个独立的处理单元就是数据项，数据元素是数据项的集合。例如，用一条记录表示一个数据元素时，这条记录中一般会有多个描述记录属性的小项，

称为数据项。通常，数据项是不可再分的数据。

(2) 原子项

若数据元素不可再分，则数据元素和数据项是同一概念，如：整数"5"、字符"A"等。

例如，例 1.1 描述一个学生信息的数据元素可由下列 6 个数据项构成：学号、姓名、性别、民族、籍贯、出生日期。其中，出生日期又可以由 3 个数据项"年"、"月"和"日"组成，因此称"出生日期"为组合项，而其他不可分割的数据项为原子项。

3. 数据对象

数据对象(Data Object)是性质相同的一类数据元素的集合，是数据的一个子集。例如，整数数据对象是 $\{0, \pm 1, \pm 2, \pm 3, \dots\}$ ，字符数据对象是 $\{a, b, c, \dots\}$ 。数据对象可以是有限的，也可以是无限的。

4. 数据结构

数据结构(Data Structure)由数据和结构两部分构成。其中，数据部分是指数据元素的集合；结构就是关系，结构部分是指数据元素之间关系的集合。所以，数据结构就是指数据元素的集合及数据元素之间关系的集合。概括地讲，数据结构就是指相互之间有一种或多种特定关系的数据元素的集合。在计算机上要处理数据，就要保存数据及它们之间的关系。在这里，关系就是数据的逻辑结构，它指反映数据元素之间的逻辑关系的数据结构，其中的逻辑关系是指数据元素之间的前后件关系，而与它们在计算机中的存储位置无关。

(1) 数据的逻辑结构

数据的逻辑结构(Logic Structure)是从具体问题抽象出来的数学模型，与数据在计算机中的具体存储没有关系。数据的逻辑结构独立于计算机，是数据本身所固有的特性。从逻辑上可以把数据结构分为线性结构和非线性结构，主要包括：集合、线性、树形和图状结构：

① 集合结构

集合结构(Set Structure)中的数据元素除了“同属于一个集合”的关系外，再无其他关系。如整数集、字符集等。

② 线性结构

线性结构(Linear Structure)中的数据元素之间存在“一对一”的关系。比如数组、队列等。

③ 树形结构

树形结构(Tree Structure)中的数据元素之间存在“一对多”的关系。比如例 1.3 中的人机对弈等。

④ 图状结构

图状结构(Graphic Structure，也称网状结构)中的数据元素之间存在“多对多”的关系。比如城市交通图等。

图 1.3 是上述几种结构的关系图。其中，树形结构和图状结构又称为非线性结构。由于集合中数据元素之间的关系是非常松散的，因此，常用其他几种结构来描述集合。

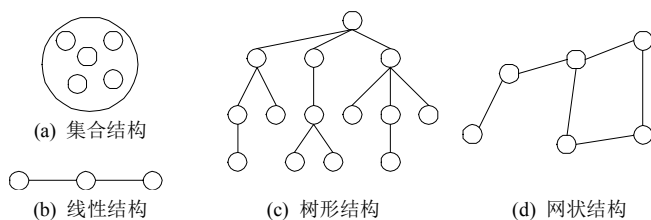


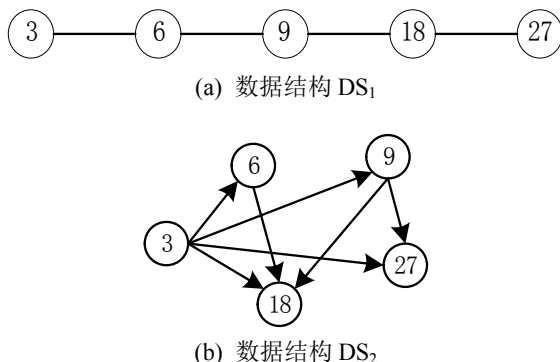
图 1.3 四种基本数据结构关系图

数据结构由相互之间存在着一种或多种关系的数据元素的集合和该集合中数据元素之间的关系组成。数据结构的形式化定义记为： $\text{Data_Structure}=(D,R)$ 。

其中， D 是数据元素的有限集合， R 是数据集合 D 中所有元素之间的关系的有限集合。

【例 1.4】 定义集合 $D=\{3, 6, 9, 18, 27\}$ 的数据结构。

$DS_1=(D,R_1)$ ，其中 R_1 定义为 D 上的“ $>$ ” (大于) 关系，则数据结构 DS_1 可以表示为如图 1.4(a)所示的形式，该结构为线性结构。 $DS_2=(D,R_2)$ ，其中 R_2 定义为 D 上的“整除”关系，则 $R_2=\{(3,6),(3,9),(3,18),(3,27),(6,18),(9,18),(9,27)\}$ ，数据结构 DS_2 可以表示为如图 1.4(b)所示的形式，该结构为图状结构。

图 1.4 集合 D 上定义的两个数据结构

从上面的例子可以看出，即使是由相同元素构成的集合，只要定义的关系不同，也不是同一数据结构。数据结构不仅描述了结构中的元素，还描述了这些元素之间的关系。数据结构的定义仅是对操作对象的一种数学描述，结构中定义的关系是数据元素之间的逻辑关系。

数据结构可以分为逻辑上的数据结构和物理上的数据结构。数据结构的形式化定义为逻辑结构。物理结构为数据在计算机中的存储，它包括数据元素的存储和关系的存储。在计算机中存储信息的最小单位是二进制的位(bit)，可以用一个由若干位组合起来形成的一个位串存储一个数据元素。因此可将位串看成是数据元素在计算机中的存储形式。

(2) 数据的物理结构(Physical Structure)

数据的物理结构，又称为存储结构(Storage Structure)，是指数据的逻辑结构在计算机存储空间中的存放形式。数据的物理结构是数据结构在计算机中的存储，它包括数据元素的机内存储和关系的机内存储。由于具体实现的方法有顺序、链式、索引、散列等多种，因此，一种数据结构可存储成一种或多种存储结构。

数据元素的机内存储方法：用二进制位的位串存储数据元素。通常称这种位串为节点

(node)。当数据元素由若干个数据项组成时，位串中与各数据项对应的子位串称为数据域(data field)。因此，节点是数据元素的机内存储。

关系的机内存储方法：数据元素之间的关系的机内存储可以分为顺序存储和非顺序存储，这样就可以得到两种不同的存储结构，即顺序存储结构和链式存储结构。顺序存储借助元素在存储器中的相对位置来存储数据元素之间的逻辑关系。链式存储借助指示元素存储位置的指针(pointer)来存储数据元素之间的逻辑关系。

① 顺序存储结构(Sequence Storage Structure)

顺序存储结构通过数据元素在计算机存储器中的相对位置来存储数据元素的逻辑关系，一般把逻辑上相邻的数据元素存储在物理位置相邻的存储单元中，它是一种最基本的存储方法，一般采用数组来实现。

② 链式存储结构(Linked Storage Structure)

链式存储结构对逻辑上相邻的两个数据元素不要求其存储位置必须相邻，元素间的逻辑关系通过指针来存储，一般采用链表来实现。链式存储结构中的数据元素称为节点(node)，在节点中附设地址域(Address Domain)来存储与该节点相邻的节点的地址，从而实现节点间的逻辑关系。图 1.5 给出了图 1.4 中数据结构 DS₁ 的不同存储方式。

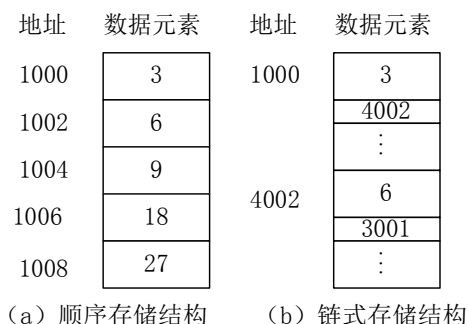


图 1.5 数据结构 DS₁ 存储结构示意图

(3) 数据的运算

算法的设计取决于数据的逻辑结构，而算法的实现依赖于数据采用的存储结构。数据的存储结构实质上是它的逻辑结构在计算机存储器中的实现，为了全面地反映一个数据的逻辑结构，它在存储器中的映象包括两方面的内容，即数据元素之间的信息和数据元素之间的关系。不同的数据结构有其相应的若干运算。数据的运算是在数据的逻辑结构上定义的操作算法，如检索、插入、删除、更新和排序等。数据的运算是数据结构的一个重要方面，讨论任何一种数据结构时都离不开对该结构上的数据运算及其实现算法的探讨。

1.3 数据类型

1.3.1 数据类型概述

数据类型(Data Type)是高级程序设计语言中的概念，是数据的取值范围和对数据进行

操作的总和。数据类型规定了程序中对象的特性。程序中的每个变量、常量或表达式的结果都应该属于某种确定的数据类型。一方面，在程序设计语言中，每一个数据都属于某种数据类型。类型显式或隐式地规定了数据的取值范围、存储方式以及允许进行的运算。数据类型是在程序设计中已经实现了的数据结构。另一方面，在程序设计过程中，当需要引入某种新的数据结构时，总是借助编程语言所提供的数据类型来描述数据的存储结构。

在客观世界中，任何数据元素都应该有自身的取值范围和所允许进行的运算操作。数据类型就是一个值的集合和定义在这个值集上的一组操作的总称。例如，C语言中的基本整数类型(signed int)，它的值集是-32768~32767，在这个值集上能进行的操作有加、减、乘、除和取余数等，而在实数类型(float)上就不能进行取余数操作。按值的不同特性，数据类型又可分为不可分解的原子类型及可分解的结构类型。比如，C语言中的整型、实型、字符型就属于原子类型；而数组、结构体和共用体类型就属于结构类型，由其他类型构造得到。

1.3.2 抽象数据类型

1. 抽象的数据类型

抽象的数据类型(Abstract Data Type, ADT)是指基于一类逻辑关系的数据类型以及定义在这个类型上的一组操作。在软件设计中，抽象数据类型通常包含元素、关系和操作三部分。所以，一般而言，抽象数据类型可用以下三元组表示：

$$\text{ADT_Type} = (D, R, P)$$

其中，D是数据元素有限集，即数据对象，R是D上的关系集，P是对D的基本操作集。

2. 抽象数据类型定义的一般形式

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>
}ADT 抽象数据类型名
```

例如，线性表这样的抽象数据类型，其数学模型是数据元素的集合，该集合内的元素有这样的关系：除第一个和最后一个外，每个元素有唯一的前趋和唯一的后继。可以有这样的一些操作：插入一个元素、删除一个元素。

例如，线性表的抽象数据类型就可以定义为：

ADT list

```
{  数据对象：任意数据元素的集合
    数据关系：除第一个和最后一个外，每个元素有唯一的直接前驱和唯一的直接后继
    基本操作：
```

```
ListInsert(&L, i, e)      /*元素的插入*/
ListDelete(&L, i, e)     /*元素的删除*/
...
}ADT list
```

3. 本书在用 C 语言描述时的约定

(1) C 语言的数组元素的下标从“0”开始，为此，在表示数据结构时，数据元素的序号也从 0 开始。

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

```
typedef int ElemType  /*定义所用数据类型为 int*/
```

(3) 数据存储结构用类型定义 typedef 描述，例如：

```
typedef struct{
    ElemType *elem;
    int length;
    int listsize;
}SqList; /*定义为 SqList 的线性表采用顺序存储结构的类型定义*/
```

(4) 算法以函数形式描述：

```
类型标识符 函数名(形参表)
/*算法说明*/
{语句}
```

通过以上定义可以看出，抽象数据类型只是数学的抽象，在 ADT 的定义中根本没有涉及如何实现操作的集合。对于每个 ADT 并不存在什么法则来说明必须要有哪些操作，这只是一个设计决策。

4. 抽象数据类型可以细分为 3 种类型

- (1) 原子类型：其值是不可分的。
- (2) 固定聚合类型：其值由确定数目的成分按某种结构组成。
- (3) 可变聚合类型：其值由不确定数目的成分构成。

一个抽象数据类型的软件模块通常包含定义、存储和实现这 3 个部分。

1.4 算法和算法复杂度

实际问题需要找出解决问题的方法。用计算机解决实际问题，就要先给出解决问题的算法，再依据算法编写程序完成要求。算法(Algorithm)是指在有限的时间范围内，为解决某一问题而采取的方法和步骤的准确完整的描述，它是一个有穷的规则序列，这些规则决定了解决某一特定问题的一系列运算。算法是程序设计的精髓，算法的设计取决于数

据的逻辑结构，算法的实现取决于数据的物理结构。

1.4.1 算法的重要性

1. 算法的五个特征

(1) 有穷性

一个算法必须总是(对任何合法的输入值)在执行有穷步之后结束，且每一步都可在有穷的时间内完成。这也是算法与程序的最主要区别，程序可以无限地循环下去，如操作系统的监控程序在机器启动后就一直监测着操作者的鼠标动作和输入的命令。

(2) 确定性

算法中的每一条指令都必须有明确的含义，不应使读者产生二义性。并且在任何条件下，算法只有唯一的一条执行路径，即对于相同的输入只能得到相同的输出。

(3) 可行性

一个算法是可以被执行的，即算法中的每个操作都可以通过已经实现的基本运算执行有限次来完成。

(4) 有输入

根据实际问题的需要，一个算法在执行时可能要接收外部数据，也可能无须外部输入。所以一个算法应有零个或多个输入，这取决于算法本身要实现的功能。

(5) 有输出

一个算法在执行完毕后，一定要有一个或多个结果或结论。这就要求算法一定要有输出，这些输出是同输入有着某些联系的量。

通常，解决同一个问题，不同的人有不同的想法，即使是同一个人，在不同的时间里可能对同一个问题的理解也不完全相同。算法是依据个人的理解和想法人为设计出来的求解问题的步骤，不同的人或同一个人不同的时间里设计出来的算法也不尽相同，那么哪种算法设计得好呢？如何评价一个算法的好与坏呢？

2. 算法效率的度量标准

通常，在算法设计时应该考虑从以下几个方面来度量算法的效率：

(1) 正确性

正确性(Correctness)，满足预先规定的功能和性能的要求，这是算法设计最基本的要求，算法应严格地按照特定的规格说明进行设计，要能够解决给定的问题。但是，“正确”一词的含义在通常的用法中有很大的区别，大体上可分为以下4个层次：

- ① 依据算法所编写的程序中不含语法错误；
- ② 程序对于几组输入数据能够得到满足规格要求说明的结果；
- ③ 程序对于经过精心挑选较为苛刻的几组输入数据也能够得到令人满意的结果；
- ④ 程序对于所有符合要求的输入数据都能得到正确的输出。

对于大型软件需要进行专业测试，一般情况下，通常以第③个要求作为衡量算法正确性的标准。

(2) 可读性

可读性(Readability)是指一个算法应当思路清晰、层次分明、简单明了、易读易懂。设计算法的主要目的是解决实际问题，在设计实现一个项目时，往往不是一个人独立完成的。为了达到可读性的要求，在设计算法时，一般要使用有一定意义的标识符来命名变量、函数等，以便于“见名知意”。其次，可以在算法的开头或指令的后面加注释来解释算法和指令的功能。

(3) 健壮性

健壮性(Robustness)是指一个算法应该具有很强的容错能力，当输入不合法的数据时，算法应当能做适当的处理，使得不至于引起严重的后果。当输入不合法的数据时，算法能做出相应的响应或进行适当的处理，避免带着非法数据执行，导致莫名其妙的结果。

(4) 高效率

运行时间(Running Time)是指算法在计算机上运行所花费的时间，它等于算法中每条语句执行时间的总和。一般来说，执行时间越短，性能越好。依据算法编写的程序运行速度较快。

(5) 低存储

占用空间(Storage Space)是指算法在计算机上存储所占用的存储空间，包括存储算法本身所占用的存储空间、算法的输入及输出数据所占用的存储空间和算法在运行过程中临时占用的存储空间。依据算法编写的程序在运行时所需内存空间较小。

对于一个系统设计人员来说，前 3 项很容易实现。在使用软件时，人们更加注重于软件的运行速度，而后两项恰恰是影响速度的主要因素。

1.4.2 时间复杂度

1. 时间复杂度的定义

一个程序的运行时间是指程序从开始到结束所需要的时间。通常，用 n 作为表示问题规模的量。我们把规模为 n 的算法的执行时间，称为时间复杂度(Time Complexity)，记为 $T(n)$ 。通常把算法中基本操作重复执行的次数(频度)作为算法的时间复杂度： $T(n)=f(n)$ 。

“渐进时间复杂度”，即当 n 逐渐增大时的极限情况。一般把这种算法的渐进复杂度简称为时间复杂度。为了便于分析，时间复杂度常用数量级的形式来表示，即 $T(n)=O(f(n))$ 。其中大写字母为 Order(数量级)的第一个字母， $f(n)$ 为函数形式，如 $T(n)=O(n^2)$ 。

一般用数量级的形式表示 $T(n)$ ，当 $T(n)$ 为多项式时，可只取其最高次幂，且其系数也可省略。算法的执行时间需通过依据该算法所编写的程序在计算机上运行时所消耗的时间来度量。

2. 度量一个程序执行时间的两种方法

(1) 事后统计法

这种统计方法依据算法所编写的程序在计算机上运行时所消耗的时间。但是，同一个程序在不同类型的机器上运行所需的时间不一定相同，所以这种统计是片面的。

(2) 事先估计法

根据每条指令的执行时间来估算依据算法编写的程序在计算机上运行时所消耗的时间。但是，因为每种类型机器的指令集不同，执行的时间也不尽相同，所以这种方法也离不开具体的机器软硬件环境和设备。

3. 时间复杂度的计算方法

显然，以具体的时间单位作为计算机程序执行时的时间变量是不科学的。所以在计算算法的执行时间时，应该抛开具体机器软硬件环境和设备，而使用指令的执行次数作为时间单位更合理一些。在算法中，可以使用基本语句的执行次数作为算法的时间度量单位，可以认为，一个特定算法时间性能只依赖于问题的规模(通常用 n 来表示)，或者说它是关于问题规模 n 的一个函数 $f(n)$ ，当问题规模 n 趋近于无穷大时的时间量级就称为算法的渐进时间复杂性，简称为时间复杂性或时间复杂度。记作 $T(n)=O(f(n))$ ，即 $T(n)$ 是 $f(n)$ 的同阶无穷大。

【例 1.5】 分析如下程序段的时间复杂度。

```
s=0;
for(i=1;i<=n;i++)
    s=s+i;
分析：
s=0;执行 1 次
i=1;执行 1 次
i<=n;执行 n+1 次
s=s+i;执行 n 次
i++;执行 n 次
```

总的执行次数为 $3(n+1)$ 次，因此，该算法的时间复杂度为 $T(n)=O(3(n+1))=O(n)$ 。

在分析算法的时间性能时，常用最基本语句的执行次数来估算。所谓最基本语句通常是指最深层循环体中的语句，也是执行频率最快的语句。它的执行次数反映了整个算法的基本时间性能。如例 1.5 中的 $s=s+i$ 和 $i++$ 均被执行了 n 次，所以 $T(n)=O(n)$ 。

4. 算法时间复杂度的时间量级

实际上，算法的时间量级有多种形式，见表 1.2，其对应的函数曲线见图 1.6。

表 1.2 算法的时间量级分类表

名称	时间复杂度 $T(n)$	说明
常量阶	$O(1)$	与问题规模无关的算法
线性阶	$O(n)$	与问题规模相关的单重循环
平方阶	$O(n^2)$	与问题规模相关的二重循环
立方阶	$O(n^3)$	与问题规模相关的三重循环
指数阶	$O(e^n)$	较为复杂
对数阶	$O(\log_2^n)$	折半查找算法
复合阶	如 $O(n \log_2^n)$	堆排序算法
其他	不太确定	过于复杂

一个算法的时间复杂度可能存在最好情况和最坏情况，通常要以算法的平均时间复杂度来进行算法分析。但是算法的平均时间复杂度取决于各种数据出现的概率，难以进行分析。所以，往往借助于最坏时间复杂度来进行算法分析与评价。

一般地，对于足够大的 n ，常用的时间复杂度存在如下顺序：

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

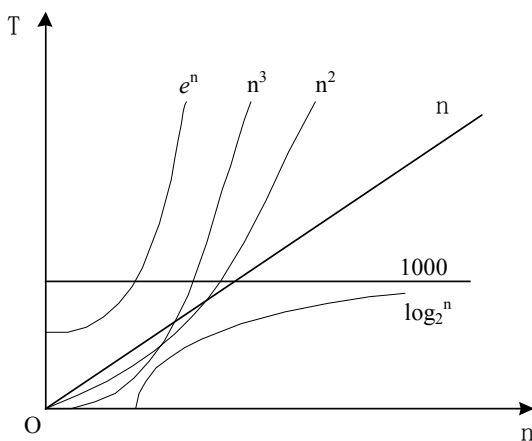


图 1.6 常见函数的增长率

1.4.3 空间复杂度

空间是指执行算法所需要的存储空间，算法所对应的程序在运行时所需的存储空间包括固定部分和可变部分。固定部分所占空间与所处理的数据结构外数据的大小和数量无关，或者称与该问题的实例的特征无关。主要包括程序代码、常量、简单变量等所占的空间；可变部分所占空间与该算法在某次执行中处理的特定数据的大小和规模有关。例如，100 个数据元素的排序算法与 1000 个数据元素的排序算法所需的存储空间显然是不同的。

1. 空间复杂度的定义

与算法的时间复杂度类似，空间复杂度也是关于问题规模 n 的一个函数，当问题规模

n 趋近于无穷大时的空间量级就称为算法的渐进空间复杂度，简称空间复杂度(Space Complexity)。可以将空间复杂度作为算法所需存储空间的度量，记作： $S(n)=O(f(n))$ 。

2. 空间复杂度的计算方法

那么，算法的空间需求有哪些呢？大体上，依据算法所编写的程序除了需要存储空间来寄存程序本身所用的指令、常数、变量和输入数据外，也需要一些对数据进行操作的工作单元和存储一些为实现计算所需信息的辅助空间。通常，程序所占空间变化不大，所以在此主要考虑算法的辅助空间需求。

【例 1.6】 分析下面程序的时间复杂度和空间复杂度。

```
RevArray(int a[],int n)
{
    int i,j,*b;
    b=(int *)malloc(sizeof(int)*n);
    for(i=0,j=n-1;i<n;i++,j--)
        b[j]=a[i];
    for(i=j=0;i<n;i++,j++)
        a[i]=b[i];
    free(b);
}
```

算法空间复杂度的分析：

因为基本语句就是循环内的赋值语句，共执行了 $2n$ 次，所以 $T(n)=2n=O(n)$ 。而算法的辅助空间是一个与问题规模同量级的一维数组空间，另外再加上两个控制变量 i 和 j ，共 $n+2$ 个，所以 $S(n)=n+2=O(n)$ 。

1.5 本章实战练习

1. 什么是数据结构？
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
3. 设有数据逻辑结构为：

```
B=(K,R)
K=(k1,k2,...,k9)
R={<k1,k2>,<k1,k8>,<k2,k3>,<k2,k4>,<k2,k5>,<k3,k9>,<k5,k6>,<k8,k9>,<k9,k7>}
```

试画出这个逻辑结构的图示。

4. 什么是抽象数据类型？
5. 什么是算法？

6. 算法应该具有什么特征?
7. 算法的衡量标准或依据是什么?
8. 分析下列程序片段的时间复杂度。

(1) 程序片段 1:

```
sum=0;
for(i=0;i<N;i++)
    sum++;
```

(2) 程序片段 2:

```
sum=0;
for(i=0;i<N;i++)
    for(j=0;j<N;j++)
        sum++;
```

(3) 程序片段 3:

```
sum=0;
for(i=0;i<N;i++)
    for(j=0;j<N*N;j++)
        sum++;
```

(4) 程序片段 4:

```
sum=0;
for(i=0;i<N;i++)
    for(j=0;j<i;j++)
        sum++;
```

(5) 程序片段 5:

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

实战练习参考答案

1. 数据结构是指同一数据元素集合中各数据元素之间存在的关系。
2. C。
3. 该逻辑结构示意图如图 1.7 所示。
4. 抽象数据类型(Abstract Data Type, ADT)是指一个数学模型以及定义在此数学模型上的一组操作, 抽象数据类型可以使人们更容易描述现实世界。
5. 算法是解题的步骤, 是指令的有限序列。
6. 一个算法应该具有 5 个特征: 有穷性、确定性、可行性、有输入、有输出。

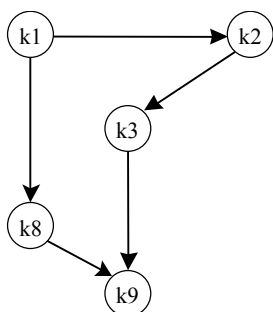


图 1.7 逻辑结构图

7. 算法是通过时间复杂度和空间复杂度进行衡量的。其中，算法的时间复杂度是指运行算法时所需要消耗的时间。算法的空间复杂度是指算法在计算机内执行时所需存储空间的度量。

8. (1) $O(n)$ 。

(2) 是嵌套循环，其时间复杂度为 $O(n^2)$ 。

(3) 关键语句 `sum++` 执行次数为 $N*N*N$ ，因此，时间复杂度为 $O(n^3)$ 。

(4) 关键语句 `sum++` 执行次数为 $1+2+3+\dots+N=N(N+1)/2$ ，因此，时间复杂度为 $O(n^2)$ 。

(5) $O(n^3)$ 。

1.6 本章小结

数据结构是计算机存储、组织数据的方式。数据结构是指相互之间存在一种或多种特定关系的数据元素的集合。一个数据结构是由数据元素依据某种逻辑关系组织起来的。对数据元素间逻辑关系的描述称为数据的逻辑结构；数据必须在计算机内存储，数据的存储结构是数据结构的实现形式，是其在计算机内的存储；此外，讨论一个数据结构必须同时讨论在该类数据结构上执行的运算才有意义。一个逻辑数据结构可以有多种存储结构，且各种存储结构会影响数据处理的效率。通常情况下，精心选择的数据结构可以带来更高的运行效率或者存储效率。

1. 数据结构指的是数据之间的相互关系，即数据的组织形式。
2. 数据元素是数据的基本单位。在不同的条件下，数据元素又可称元素、节点、顶点、记录等。
3. 数据对象是性质相同的数据元素的集合，是数据的一个子集。
4. 数据结构的 4 种基本结构是集合结构、线性结构、树形结构、图状结构。
5. 数据结构的定义形式为

`Data_Structure=(D,R)`

其中， D 是数据元素的有限集， R 是 D 上关系的有限集。

6. 抽象数据类型是指一个数学模型以及定义在此数学模型上的一组操作。抽象数据类型型可以使我们更容易描述现实世界。

- ## 1.7 习题 1

1. 算法分析的目的是()。
A、找出数据结构的合理性
B、研究算法中的输入和输出的关系
C、分析算法的效率以求改进
D、分析算法的易懂性和文档性
2. 数据结构中,与所使用的计算机无关的是数据的()结构。
A、存储
B、物理
C、逻辑
D、物理和存储
3. 对一个算法的评价,不包括如下()方面的内容。
A、健壮性和可读性
B、并行性
C、正确性
D、时空复杂度
4. 算法分析的两个主要方面是()。
A、空间复杂性和时间复杂性
B、正确性和简明性
C、可读性和文档性
D、数据复杂性和程序复杂性
5. 算法指的是()。
A、计算机程序
B、解决问题的计算方法
C、排序算法
D、解决问题的有限运算序列
6. 线性结构表示数据元素之间存在一种()关系。
A、一对多
B、多对多
C、多对一
D、一对一
7. 某程序的时间复杂度为 $(3n+n\log_2^n+n^2+8)$,其数量级表示为()。
A、 $O(n)$
B、 $O(n\log_2^n)$
C、 $O(n^2)$
D、 $O(\log_2^n)$
8. 计算机算法指的是()。
A、计算方法
B、排序方法
C、解决问题的有限运算序列
D、调度方法
9. 计算机算法必须具备输入、输出和()等 5 个特性。
A、可行性、可移植性和可扩充性
B、可行性、确定性和有穷性
C、确定性、有穷性和稳定性
D、易读性、稳定性和安全性
10. 以下属于逻辑结构的是()。
A、顺序表
B、哈希表
C、有序表
D、单链表

二、填空题

1. 数据结构按逻辑结构可分为两大类, 它们分别是_____和_____。
2. 数据的物理结构主要包括_____和_____两种情况。
3. 数据结构是指数据及其相互之间的_____。当节点之间存在 M 对 N(M: N)的关系时, 称这种结构为_____。
4. 数据的逻辑结构被分为_____、线性结构、树形结构和_____4 种。
5. 线性结构中元素之间存在_____关系, 树形结构中元素之间存在_____关系。
6. 数据结构的定义形式为(D,R), 其中 D 是_____的有限集合, R 是 D 上_____的有限集合。
7. 在图状结构中, 每个节点的前驱节点数和_____节点数可以_____。
8. 抽象数据类型包括_____和_____两个部分。
9. 在非数值计算的程序设计问题中, 数据结构是一门研究计算机的_____及相互之间的_____和运算等的学科。
10. 在线性结构中, 第一个节点_____前驱节点, 其余每个节点有且只有 1 个前驱节点; 最后一个节点_____后继节点, 其余每个节点有且只有 1 个后继节点。
11. 数据的运算最常用的有 5 种, 它们分别是插入、_____、修改、_____和排序。
12. 数据结构包括数据的_____、数据的_____和数据的运算这三个方面的内容。
13. 在树形结构中, 根节点没有_____节点, 其余每个节点有且只有_____个前驱节点。
14. 数据的存储结构可用 4 种基本的存储方法表示, 它们分别是_____、_____、链式和散列。

三、判断题

1. 算法的优劣与算法描述语言无关, 但与所用计算机有关。()
2. 数据的物理结构是指数据在计算机内的实际存储形式。()
3. 数据元素是数据的最小单位。()
4. 数据的逻辑结构是指数据的各数据项之间的逻辑关系。()

四、简答题

1. 分析下面程序段的时间复杂度。

```
x=0;
for(i=1; i<n; i++)
    for(j=1; j<=n-i; j++)
        x++;
```

2. 分析下面程序段的时间复杂度。

```
i=1;  
while(i<=n) i=i*3;
```

3. 分析下面程序段的时间复杂度。

```
s=0;  
for(i=0; i<n; i++)  
    for(j=0; j<n; j++)  
        s+=B[i][j];  
sum=s;
```

4. 分析下面程序段的时间复杂度。

```
for(i=0;i<n;i++)  
    for(j=0;j<m;j++)  
        a[i][j]=0;)
```