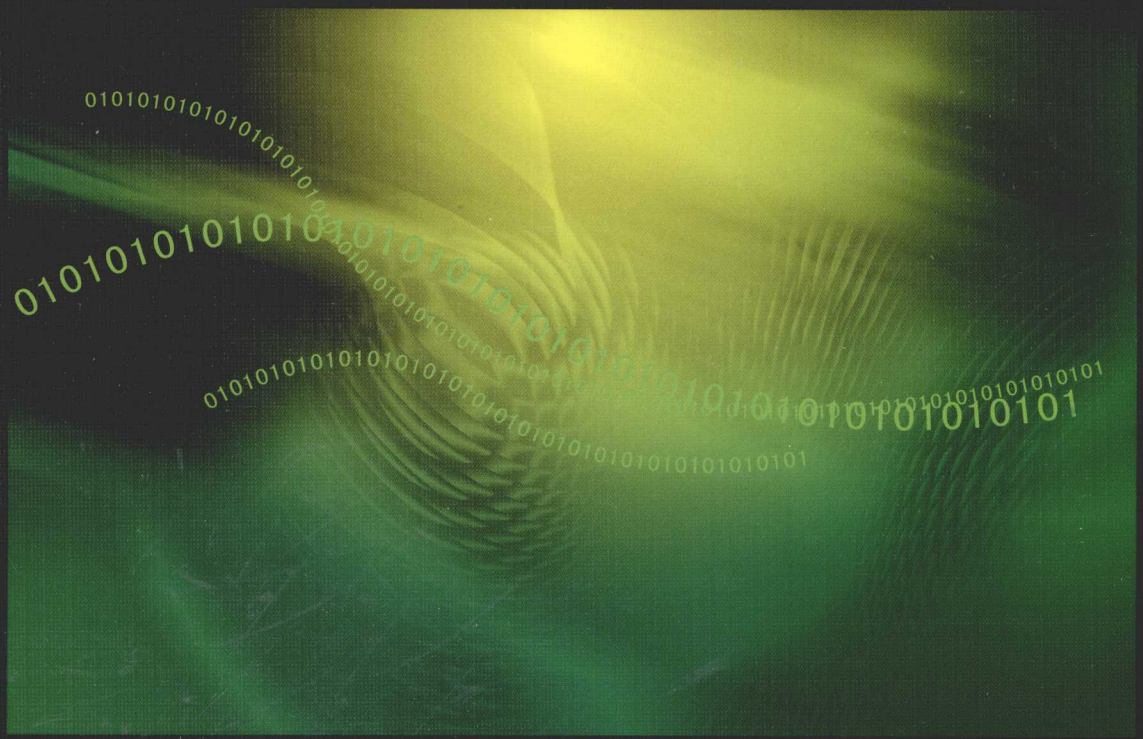
21世纪

普通高等教育电气信息类

应用型规划教材

实用数据结构

侯 虹 文玉锋 编著



化学工业出版社

21世纪普通高等教育电气信息类应用型规划教材

实用数据结构

侯　虹　文玉锋　编著

化学工业出版社

·北京·

全书共分为9章，主要内容包括：数据结构基本概念与算法分析，线性表，栈和队列，串，数组、特殊矩阵和广义表，树和二叉树，图，查找，内部排序方法。本书内容编排由浅入深、循序渐进，每章最后配有相应的习题和答案用来巩固所学的理论知识，书末附有作者在几年授课过程中带学生在 TruboC 2.0 环境下调试通过的一些程序供参考，既便于教学，又便于自学。

本书可作为计算机类专业或信息管理类相关专业的本科、高职高专教材，也可供从事计算机工程与应用工作的科技工作者参考。

图书在版编目（CIP）数据

实用数据结构 / 侯虹，文玉峰编著. —北京：化学工业出版社，2010.6

21世纪普通高等教育电气信息类应用型规划教材

ISBN 978-7-122-08356-2

Ⅰ. 实… Ⅱ. ①侯… ②文… Ⅲ. 数据结构-高等学校-教材 Ⅳ. TP311.12

中国版本图书馆 CIP 数据核字（2010）第 073982 号

责任编辑：唐旭华　宋湘玲　　文字编辑：孙　科

责任校对：王素芹　　装帧设计：尹琳琳

出版发行：化学工业出版社（北京市东城区青年湖南街 13 号　邮政编码 100011）

印　　装：三河市延风印装厂

787mm×1092mm　1/16　印张 12¾　字数 327 千字　　2010 年 8 月北京第 1 版第 1 次印刷

购书咨询：010-64518888（传真：010-64519686）　售后服务：010-64518899

网　　址：http: // www. cip. com. cn

凡购买本书，如有缺损质量问题，本社销售中心负责调换。

定　　价：26.00 元

前　言

编写本书的原因非常简单：帮助刚学完C语言的学生建立正确的编程思维方式。从事计算机研究和开发的人都应该很清楚，语言只是一种工具，算法才是灵魂。面对一个复杂的具体问题，将之抽象成一个数学模型，学会剖析数据的逻辑结构，选择合适的存储方式，有效地描述和组织数据，利用数据结构的基本操作原理去设计程序的算法，然后动手编辑、调试程序，最终达到问题的解决。不少学生自认为C语言学得还可以，但是程序编写缺乏合理的编程规格和风范，不明白程序设计不仅在于设计计算机编程语言，而且在于如何有效描述、组织和操作数据。还有的学生甚至学完数据结构以后也没有感觉到“要成为一名优秀的专业软件开发人员，不学或没学好数据结构是不可想象的事情”这句话的真谛。“数据结构”其实是一门理论性极强的计算机类专业基础课程，但是本书的编写尽量避开太生僻的理论研究，内容编排由浅入深、循序渐进，每章最后配有相应的习题和答案用来巩固所学的理论知识，书的最后附有笔者在几年授课过程中带学生在TruboC 2.0环境下调试通过的一些程序，仅供大家参考，既便于教学，又便于自学。

全书分为9章，第1章是数据结构绪论，重点讲解了什么是数据结构，数据结构的基本概念和算法的分析与评价；第2章介绍线性表，分析了线性表的逻辑结构、不同的存储结构及在不同存储方式下基本运算实现的方法；第3章介绍了栈和队列，它们是线性表的内涵引申和应用特例；第4章是串，主要讲解了串的基本运算及模式匹配；第5章介绍了数组、特殊矩阵和广义表，重点是特殊矩阵、稀疏矩阵的存储和广义表的基本操作的实现；第6章是关于树和二叉树的内容，主要讲解了二叉树的性质、遍历、线索化以及应用，还有赫夫曼树的概念及应用；第7章介绍了图的基本概念、存储方式、图的遍历、图的最小生成树算法、最短路径的算法、AOV网和AOE网；第8章是查找，按照线性表的查找、树表查找、哈希表的查找分别展开讲解。第9章介绍了常用的内部排序方法，它们分别是：插入排序、交换排序、选择排序、归并排序和基数排序。

尽管不同数据结构的数据操作存在差异性，但笔者认为，任何数据结构在结点的插入、修改、查询、删除等方面的操作思路却是一样的，只要认真理解一个，就可以一通百通。对学生的最低要求是学会调用，最高要求是知其然还要知其所以然，真正掌握知识的要点。本书可作为计算机类专业或信息管理类相关专业的本科或高职高专教材，也可供从事计算机工程与应用工作的科技工作者参考。

本书相关电子课件可免费提供给采用本书作为教材的大专院校使用，如有需要可发邮件至txh@cip.com.cn索取。

本书由北京化工大学侯虹、西北师范大学文玉锋编著。在编写过程中，得到了张建青、陆士桓、赵雪梅、马亚丽等各位同仁的大力支持及帮助，在此表示由衷的感谢！

由于编者水平有限，难免存在不足之处，热切期望专家和读者批评指正。

编　者

2010年5月

目　录

第 1 章　绪论 ……1
1.1　数据结构的定义 ……1
1.2　有关概念和术语 ……2
1.3　抽象数据类型 ……3
1.3.1　数据类型 ……3
1.3.2　抽象数据类型 ……3
1.4　算法和算法分析 ……3
1.4.1　算法及特性 ……3
1.4.2　算法设计的要求 ……4
1.4.3　算法描述 ……4
1.4.4　算法性能分析与评价 ……4
习题 1 ……5
第 2 章　线性表 ……7
2.1　线性表的逻辑结构 ……7
2.1.1　线性表的定义 ……7
2.1.2　线性表的基本操作 ……7
2.2　线性表的顺序表示和实现 ……8
2.2.1　顺序表 ……8
2.2.2　顺序表基本运算的实现 ……8
2.2.3　顺序表应用举例 ……10
2.3　线性表的链式表示和实现 ……10
2.3.1　单链表 ……11
2.3.2　单链表上基本运算的实现 ……11
2.3.3　循环链表 ……15
2.3.4　双向链表 ……15
2.3.5　单链表应用举例 ……16
2.4　顺序表和链表的比较 ……18
习题 2 ……19
第 3 章　栈和队列 ……22
3.1　栈 ……22
3.1.1　栈的定义及基本运算 ……22
3.1.2　栈的存储结构 ……22
3.2　栈的应用举例 ……25
3.3　队列 ……29
3.3.1　队列的定义及基本运算 ……29
3.3.2　队列的存储结构及运算 ……30
3.4　队列应用举例 ……34
习题 3 ……35
第 4 章　串 ……40
4.1　串及基本运算 ……40
4.1.1　串的基本概念 ……40
4.1.2　串的基本运算 ……40
4.2　串的定长顺序存储及基本运算 ……41
4.2.1　串的定长顺序存储 ……41
4.2.2　定长顺序串的基本运算 ……42
4.2.3　模式匹配 ……43
习题 4 ……47
第 5 章　数组、特殊矩阵和广义表 ……49
5.1　多维数组 ……49
5.1.1　数组的概念 ……49
5.1.2　数组的存储结构 ……49
5.2　特殊矩阵及压缩存储 ……50
5.2.1　对称矩阵 ……50
5.2.2　三角矩阵 ……51
5.2.3　带状矩阵 ……52
5.3　稀疏矩阵 ……53
5.3.1　稀疏矩阵的三元组表存储 ……53
5.3.2　稀疏矩阵的十字链表存储 ……56
5.4　广义表 ……58
5.4.1　广义表的定义和基本运算 ……58
5.4.2　广义表的存储结构 ……59
5.4.3　广义表基本操作的实现 ……61
习题 5 ……64
第 6 章　树和二叉树 ……66
6.1　树的概念与定义 ……66
6.1.1　树的定义 ……66
6.1.2　树的相关概念 ……66
6.2　二叉树 ……67
6.2.1　二叉树的定义 ……67
6.2.2　二叉树的主要性质 ……67
6.3　二叉树的存储结构与基本操作 ……69
6.3.1　二叉树的存储 ……69
6.3.2　二叉树的基本操作及实现 ……70
6.4　二叉树的遍历 ……72

6.4.1 二叉树的遍历方法及递归实现 ……72
6.4.2 由遍历序列恢复二叉树 ……76
6.4.3 二叉树遍历算法的应用 ……77
6.5 线索二叉树 ……79
6.5.1 线索二叉树的定义及结构 ……79
6.5.2 线索二叉树的基本操作实现 ……80
6.6 赫夫曼树及应用 ……82
6.6.1 赫夫曼树的基本概念 ……82
6.6.2 赫夫曼树在编码问题中的应用 ……83
习题 6 ……85
第 7 章 图 ……88
7.1 图的基本概念 ……88
7.1.1 图的定义 ……88
7.1.2 图的相关术语 ……88
7.2 图的存储表示 ……90
7.2.1 邻接矩阵 ……90
7.2.2 邻接表 ……91
7.3 图的遍历 ……93
7.3.1 深度优先搜索 ……93
7.3.2 广度优先搜索 ……94
7.4 生成树和最小生成树 ……96
7.4.1 生成树的概念 ……96
7.4.2 Prim 算法 ……96
7.4.3 Kruskal 算法 ……98
7.5 最短路径 ……99
7.5.1 从一个源点到其他各点的最短路径 ……99
7.5.2 每一对顶点之间的最短路径 ……102
7.6 有向无环图及其应用 ……103
7.6.1 AOV 网与拓扑排序 ……103
7.6.2 AOE 网与关键路径 ……105
习题 7 ……109
第 8 章 查找 ……116
8.1 基本概念与术语 ……116
8.2 基于线性表的查找 ……117
8.2.1 顺序查找法 ……117
8.2.2 折半查找 ……118
8.2.3 分块查找 ……120
8.3 基于树的查找 ……121
8.3.1 二叉排序树 ……121
8.3.2 平衡二叉树 ……125
8.4 哈希表查找 ……129
8.4.1 哈希函数的构造 ……130
8.4.2 处理冲突的方法 ……131
8.4.3 哈希表的查找过程及性能分析 ……132
习题 8 ……133
第 9 章 内部排序 ……136
9.1 基本概念 ……136
9.2 插入排序 ……137
9.2.1 直接插入排序 ……137
9.2.2 折半插入排序 ……137
9.2.3 希尔排序 ……138
9.3 交换排序 ……139
9.3.1 冒泡排序 ……139
9.3.2 快速排序 ……140
9.4 选择排序 ……142
9.4.1 直接选择排序 ……142
9.4.2 堆排序 ……143
9.5 归并排序 ……145
9.6 基数排序 ……146
9.6.1 多关键码排序 ……147
9.6.2 链式基数排序 ……147
9.7 各种内部排序方法的比较 ……150
习题 9 ……150
习题参考答案 ……153
附录 ……170
程序 1 顺序表的运算 ……170
程序 2 单链表的运算 ……172
程序 3 两个栈共享空间 ……174
程序 4 循环队列的入队和出队 ……176
程序 5 链队列的入队和出队 ……178
程序 6 数制转换 ……180
程序 7 二叉树的建立及中序遍历（递归） ……182
程序 8 二叉树的建立及中序遍历（非递归） ……184
程序 9 二叉查找树的静态查找 ……187
程序 10 二叉查找树（二叉排序树）的建立及遍历 ……189
程序 11 折半查找 ……191
程序 12 直接插入排序和冒泡排序 ……193
参考文献 ……195

第1章 绪 论

数据结构是从事计算机教学、研究、开发、应用等工作所必须学习和掌握的一门基础课程，它专门研究由现实世界抽象出来的数据在计算机系统中的表示、组织、存储和处理方法。用计算机存储数据不仅要存储数据本身，还要存储数据之间的各种联系——数据结构。本章主要介绍与数据结构有关的概念和术语，通过本章学习，读者应能掌握数据、数据元素、逻辑结构、存储结构、数据处理、算法设计等基本概念，并了解如何评价一个算法的好与坏。

1.1 数据结构的定义

什么是数据结构？以下通过分析利用计算机解决一个具体问题时需要做的工作来理解数据结构的概念。用计算机解决一个具体问题时要考虑以下步骤。

① 从具体问题中抽象出一个适当的数学模型。即从具体问题中找出操作对象之间的关系，然后用数学语言加以描述。

② 设计一个适合该数学模型的算法。

③ 编写程序。

④ 对程序进行测试、调整、修改，直至最终解决问题。

在实际问题中，各个数据元素之间的关系有线性的、树形的和网状的等。

- 线性关系：数据元素之间存在一对一的线性关系。

如列车中各车厢之间的关系是线性的；排队买车票时人与人之间的关系是线性的，诸如此类关系的还有查号系统自动化、考试查分系统、仓库库存管理系统等。在这类问题中，计算机处理的对象之间通常存在着一种简单的线性关系，这类数学模型称为线性数据结构。

- 树形关系：元素之间存在着一对多的关系。

如八皇后问题就是典型的树形关系。在八皇后问题中，处理过程不是根据某种确定的计算法则，而是利用试探和回溯的探索技术求解。为了求得合理布局，在计算机中要存储布局的当前状态。从最初的布局状态开始，一步步地进行试探，每试探一步形成一个新的状态，整个试探过程形成了一棵隐含的状态树。此问题中所出现的树也是一种数据结构，其可以应用在许多非数值计算的问题中。

- 网状关系：元素之间存在多对多的关系。

如教学计划编排问题就是典型的网状关系。一个教学计划包含许多课程，在教学计划包含的许多课程之间，有些必须按规定的先后次序进行，有些则没有次序要求。即有些课程之间有先修和后修的关系，有些课程可以任意安排次序。这种各个课程之间的次序关系可用网来表示。

综上所述，描述这类非数值计算问题的数学模型不再是数学方程，而需要建立诸如表、树、图之类的数据结构，来描述所处理对象的特性以及各对象之间的关系。因此，**数据结构**是一门研究非数值计算的程序设计问题中的计算机操作对象以及它们之间的关系和操作的学科。

1.2 有关概念和术语

数据（Data）是计算机表示客观事物的载体，它能够被计算机识别、存储、加工和处理。计算机科学中，所谓数据就是计算机加工处理的对象，它可以是数值数据，也可以是非数值数据。数值数据是一些整数、实数或复数，主要用于工程计算、科学计算和商务处理等；非数值数据包括字符、文字、图形、图像、语音等。

数据元素（Data Element）是数据的基本单位。数据的范畴非常广泛，数据元素也有大有小，在不同的条件下，数据元素又可称为元素、结点、顶点、记录等。例如，学生信息检索系统中学生信息表中的一个记录、八皇后问题中状态树的一个状态、教学计划编排问题中的一个顶点等，都被称为一个数据元素。有时，一个数据元素可由若干个数据项（Data Item）组成，例如，学籍管理系统中学生信息表的每一个数据元素就是一个学生记录。它包括学生的学号、姓名、性别、籍贯、出生年月、成绩等数据项。

数据结构（Data Structure）是指互相之间存在着一种或多种关系的数据元素的集合。在任何问题中，数据元素之间都不会是孤立的，在它们之间都存在着这样或那样的关系，这种数据元素之间的关系称为结构。根据数据元素间关系的不同特性，通常有下列四类基本的结构（图 1.1）。

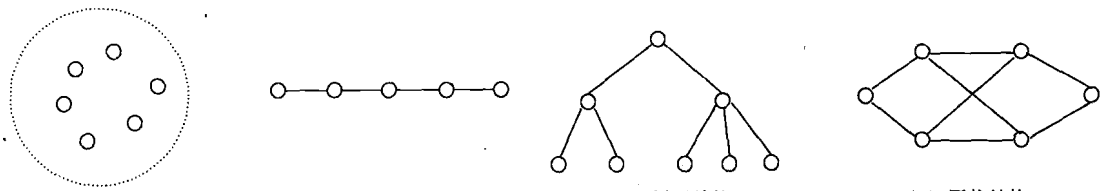


图 1.1 四类基本结构的示意图

① 集合结构：在集合结构中，数据元素间的关系是“属于同一个集合”。集合是元素关系极为松散的一种结构。

② 线性结构：该结构的数据元素之间存在着一对一的关系。

③ 树形结构：该结构的数据元素之间存在着一对多的关系。

④ 网状结构：该结构的数据元素之间存在着多对多的关系，网状结构也称作图形结构。

从数据结构的概念中可以看出，一个数据结构有两个要素：一个是数据元素的集合，另一个是关系的集合。在形式上，数据结构通常可以采用一个二元组来表示。

数据结构的形式定义为

$$\text{Data\_Structure} = (D, R)$$

其中，D 是数据元素的有限集，R 是 D 上关系的有限集。

数据结构包括数据的逻辑结构和数据的物理结构。数据的逻辑结构可以看作是从具体问题抽象出来的数学模型，它与数据的存储无关。研究数据结构的目的是为了在计算机中实现对它的操作，为此还需要研究如何在计算机中表示一个数据结构。数据结构在计算机中的标识（又称映象）称为数据的物理结构，或称存储结构。它所研究的是数据结构在计算机中的实现方法，包括数据结构中元素的表示及元素间关系的表示。

数据的存储结构可采用顺序存储或链式存储的方法。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。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。

1.3 抽象数据类型

1.3.1 数据类型

数据类型（Data Type）是和数据结构密切相关的一个概念。在用高级语言编写的程序中，每个变量、常量或表达式都有一个确定的数据类型。类型规定了在程序执行过程中变量或表达式所有可能的取值范围，以及在这些值上所允许进行的操作。可见，数据类型是一个值的集合和定义在这个集合上的一组操作的总称。

高级程序设计语言中的数据类型可分为原子类型和结构类型。原子类型的值是不可分解的。C 语言中的基本类型（int,float,char 等）、指针类型、空类型都是原子类型。结构类型是由若干类型组合而成的，是可以分解的。C 语言中的结构类型和数组类型都是由其他类型定义的。

在计算机中，数据类型并非局限于高级语言中的一个具体类型，所以用抽象数据类型表示类型。在上机实现时，再把抽象数据类型用具体的类型代替。

1.3.2 抽象数据类型

抽象数据类型（Abstract Data Type，ADT）是指一个数学模型以及定义在该模型上的一组操作。抽象数据类型的定义取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关。即不论其内部结构如何变化，只要它的数学特性不变，都不影响其外部的使用。

抽象数据类型和数据类型实质上是一个概念。例如，各种计算机都拥有的整数类型就是一个抽象数据类型，尽管它们在不同处理器上的实现方法可以不同，但由于其定义的数学特性相同，在用户看来都是相同的。因此，“抽象”的意义在于数据类型的数学抽象特性。

但在另一方面，抽象数据类型的范畴更广，它不再局限于前述各处理器中已定义并实现的数据类型，还包括用户在设计软件系统时自己定义的数据类型。为了提高软件的重用性，在近代程序设计方法学中，要求在构成软件系统的每个相对独立的模块上，定义一组数据和施于这些数据上的一组操作，并在模块的内部给出这些数据的表示及其操作的细节，而在模块的外部使用的只是抽象的数据及抽象的操作。这也就是面向对象的程序设计方法。

抽象数据类型的定义可以由一种数据结构和定义在其上的一组操作组成，而数据结构又包括数据元素及元素间的关系，因此抽象数据类型一般可以由元素、关系及操作三种要素来定义。

1.4 算法和算法分析

1.4.1 算法及特性

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

① 有穷性。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。算法的每一步必须有确切的定义，无二义性。算法的执行对应着的相同的输入仅有唯一的一条路经。

③ 可行性。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。

⑤ 输出。一个算法具有一个或多个输出。

算法的含义与程序十分相似，但又有区别。算法与数据结构是相辅相成的。解决某一特定类型问题的算法可以选定不同的数据结构，而且选择恰当与否直接影响算法的效率。反之，一种数据结构的优劣由各种算法的执行来体现。

1.4.2 算法设计的要求

要设计一个好的算法通常要考虑以下几个基本要求。

① 正确性。对于一切合法的输入，算法的执行结果都应当满足预先规定的功能和性能要求。

② 可读性。一个算法应当思路清晰、层次分明、简单明了、易读易懂，有助于人对算法的理解。

③ 健壮性。当输入数据不合法时，算法应能作适当处理，而不会产生莫明其妙的输出结果。

④ 高效率和低存储量。算法的效率通常是指算法的执行时间。对于一个具体问题的解决通常可以有多个算法，执行时间短的算法其效率就高。存储量需求是指算法在执行过程中需要的存储空间，存储需求低的算法优于存储需求高的算法。效率与存储需求都和问题的规模有关。

1.4.3 算法描述

为了表示一个算法，可以用不同的方法，常用的方法有以下几种。

① 自然语言。用自然语言来描述算法的优点是简单且便于人们对算法的阅读。缺点是不够严谨，文字冗长，容易出现歧义。

② 流程图表示。流程图是用一些图框来表示各种操作。这种方法表示算法直观形象，易于理解。这种流程图中用的篇幅较多，尤其当程序较为复杂时，画流程图既费时又不方便。

③ N-S 图表示。N-S 图表示算法比文字描述直观、形象、易于理解；比流程图紧凑易画。

④ 用伪代码表示算法。为了解决理解与执行这两者之间的矛盾，伪代码用介于自然语言和计算机语言之间的文字和符号来描述算法。不用图形符号，书写方便，格式紧凑，容易理解，便于向计算机语言算法过渡。

1.4.4 算法性能分析与评价

一种数据结构的优劣由实现其各种运算的算法具体体现，对数据结构的分析实质上就是对实现运算算法的分析，除了要验证算法是否正确解决该问题之外，还需要对算法的效率作出性能分析。评价算法的标准很多，主要看该算法所占机器资源的多少，而这些资源中时间代价和空间代价是两个最主要的方面，通常以算法执行所需要的机器时间和所占用的存储空间来评价一个算法的优劣。

将一个算法转换成程序并在计算机上执行时，其运行所需要的时间取决于硬件、书写程序的语言、目标代码的质量以及问题规模等因素。在各种因素都不能确定的情况下，很难比

较出算法的执行时间。也就是说，使用执行算法的绝对时间来衡量算法的效率是不合适的。撇开这些与计算机硬件、软件有关的因素，一个特定算法的运行工作量的大小就只依赖于问题的规模（通常用正整数 n 表示），或者说它是问题规模的函数。

（1）时间复杂度

算法的**时间复杂度**（**Time Complexity**）是指程序运行从开始到结束所需要的时间。

一个算法是由所采用的控制结构和原操作构成的，其执行时间取决于两者的综合效果。为了便于比较同一问题的不同的算法，通常的做法是：从算法中选取一种对于所研究的问题来说是基本运算的原操作，以该原操作重复执行的次数作为算法的时间度量。一般情况下，算法中原操作重复执行的次数是规模 n 的某个函数 $T(n)$。

许多时候要精确地计算 $T(n)$是困难的，引入渐进时间复杂度在数量上估计一个算法的执行时间，也能够达到分析算法的目的。

定义：如果存在两个正常数 c 和 n_0，使得对所有的 n，$n \geqslant n_0$，有

$$f(n) \leqslant cg(n)$$

则有

$$f(n) = O[g(n)]$$

例如，一个程序的实际执行时间为 $T(n)=2.7n^3+3.8n^2+5.3$，则 $T(n)=O(n^3)$。

用此方法表示的算法的时间复杂度，称为**算法的渐进时间复杂度**（**Asymptotic Complexity**）。

通常用 $O(1)$表示常数计算时间。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

算法的**空间复杂度**（**Space Complexity**）是指程序运行从开始到结束所需的存储量。

程序的一次运行是针对所求解的问题的某一特定实例而言的。例如，求解排序问题的排序算法的每次执行是对一组特定个数的元素进行排序。对该组元素的排序是排序问题的一个实例。元素个数可视为该实例的特征。

程序运行所需的存储空间包括以下两部分。

① 固定部分。这部分空间与所处理数据的大小和个数无关，或者说与问题的实例的特征无关。主要包括程序代码、常量、简单变量、定长成分的结构变量所占的空间。

② 可变部分。这部分空间大小与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。

算法的执行时间和对存储空间的耗费两者是相互矛盾的，难以兼得，即算法执行时间上的节省一定是以增加存储空间为代价的，反之亦然。一般情况下，常以算法执行时间作为算法优劣的主要衡量指标。

习题 1

一、填空题

1．数据结构被定义为（D, R），其中 D 是_____的有限集合，R 是 D 上的_____有限集合。

2．数据结构按逻辑结构可分为两大类，它们分别是_____和_____。

3．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，图形结构中元素之间存在_____关系。

4．一个算法的效率可分为_____效率和_____效率。

5．简单地说，一个算法所进行的计算次数的多少称为_____，一个算法所需要辅助存储空间的多少

称为_____。

6. 根据数据元素之间关系的不同特性，通常有四类基本结构，它们是集合、_____、_____、_____。

二、选择题

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. 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
3. 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
4. 计算机算法必须具备输入、输出和_____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
5. 数据元素是数据的基本单位，其内_____数据项。
   A. 只能包含一个　　B. 不包含　　C. 可以包含多个　　D. 必须包含多个
6. 逻辑关系是指数据元素间的_____。
   A. 类型　　B. 存储方式　　C. 结构　　D. 数据项
7. 数据结构有_____种基本逻辑结构。
   A. 1　　B. 2　　C. 3　　D. 4
8. 下列四种基本的逻辑结构中，数据元素之间关系最弱的是_____。
   A. 集合　　B. 线性结构　　C. 树形结构　　D. 图状结构
9. 一个存储结构点存储一个_____。
   A. 数据项　　B. 数据元素　　C. 数据结构　　D. 数据类型
10. 某算法的时间复杂度为 $O(2^n)$，表明该算法的_____。
   A. 问题规模是 2^n　　B. 执行时间等于 2^n
   C. 执行时间与 2^n 成正比　　D. 问题规模与 2^n 成正比
11. 算法执行时间一般与_____无关。
   A. 问题规模大小　　B. 计算机的档次
   C. 程序设计语言的种类或版本　　D. 算法设计者的水平
12. 算法分析的主要任务是分析算法_____。
   A. 是否具有较好的可读性　　B. 是否存在语法错误
   C. 功能是否符合设计要求　　D. 执行时间和问题规模之间的关系。
13. 下列时间复杂度中最坏的是_____。
   A. $O(1)$　　B. $O(n)$　　C. $O(\log_2^n)$　　D. $O(n^2)$
14. 下列算法的时间复杂度是_____。

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

   A. $O(1)$　　B. $O(n)$　　C. $O(\log_2^n)$　　D. $O(n^2)$
15. 算法的可读性是指_____。
   A. 算法所含语句数较少　　B. 算法较简单，计算机容易编译
   C. 算法较简单，很容易看出它的执行结果　　D. 算法结构清晰，容易被算法设计者及其他人看懂

三、简答题

1. 数据结构中有哪几类数据结构？哪种关系最简单？哪种关系最复杂？
2. 简述顺序存储结构与链式存储结构在表示数据元素之间关系上的主要区别。
3. 数据与数据元素有何区别？二者有什么关系？

第2章 线 性 表

线性表是最简单、最常用的一种数据结构。这种结构有以下特点：存在一个唯一没有前驱的数据元素；存在一个唯一没有后继的数据元素；此外，每个元素都有且仅有一个直接前驱和直接后继。它有两种存储方法：顺序存储和链式存储，其主要基本操作有插入、删除和查询等。通过本章学习，读者应能掌握线性表的逻辑结构和存储结构，以及线性表的基本运算和实现算法。

2.1 线性表的逻辑结构

2.1.1 线性表的定义

线性表是由 n 个类型相同的数据元素构成的有限序列，数据元素之间是一对一的关系，即每个元素有且仅有一个直接前驱和一个直接后继（除第一个元素和最后一个元素之外）。例如学生情况信息表是一个线性表；一个字符串也是一个线性表等。线性表定义如下：

线性表是具有相同数据类型的 $n(n\geqslant 0)$个数据元素的有限序列，通常记为

$$(a_1,\ a_2,\ \cdots\ a_{i-1},\ a_i,\ a_{i+1},\ \cdots,\ a_n)$$

其中，n 为表长，$n=0$ 时称为空表。表中相邻元素之间存在着序偶关系。将 a_{i-1} 称为 a_i 的直接前趋（$2\leqslant i\leqslant n$），a_{i+1} 称为 a_i 的直接后继（$1\leqslant i\leqslant n-1$）。对于 a_i，当 $i=2$，…，n 时，有且仅有一个直接前趋 a_{i-1}，当 $i=1$，2，…，$n-1$ 时，有且仅有一个直接后继 a_{i+1}，而 a_1 是表中第一个元素，它没有前趋，a_n 是最后一个元素，无直接后继。

2.1.2 线性表的基本操作

线性表是一个相当灵活的数据结构，它的长度可根据需要增长或缩短，对线性表的数据元素不仅可以进行访问，还可以进行插入和删除等。线性表上的基本操作如下。

① 线性表初始化：Init_List(L)

操作结果：构造一个空的线性表。

② 求线性表的长度：Length_List(L)

操作结果：返回线性表中的所含元素的个数。

③ 取表元：Get_List(L,i)

操作结果：返回线性表 L 中的第 i 个元素的值或地址。

④ 按值查找：Locate_List(L,x)， x 是给定的一个数据元素。

操作结果：在表 L 中查找值为 x 的数据元素，其结果返回在 L 中首次出现的值为 x 的那个元素的序号或地址，称为查找成功；否则，在 L 中未找到值为 x 的数据元素，返回一特殊值表示查找失败。

⑤ 插入操作：Insert_List(L,i,x)

操作结果：在线性表 L 的第 i 个位置上插入一个值为 x 的新元素，这样使原序号为 $i,i+1,\cdots,n$ 的数据元素的序号变为 $i+1,i+2,\cdots,n+1$，插入后表长增加 1。

⑥ 删除操作：Delete_List(L,i)

操作结果：在线性表 L 中删除序号为 i 的数据元素，删除后使序号为 $i+1,i+2,\cdots,n$ 的元

素变为序号为 $i, i+1, \cdots, n-1$，新表长减少 1。

2.2 线性表的顺序表示和实现

2.2.1 顺序表

线性表的顺序存储是指在内存中用地址连续的一块存储空间顺序存放线性表的各元素，用这种存储形式存储的线性表称为顺序表。设 a_1 的存储地址为 $\mathrm{Loc}(a_1)$，每个数据元素占 k 个存储单元，则第 i 个数据元素的地址为

$$\mathrm{Loc}(a_i)=\mathrm{Loc}(a_1)+(i-1)k \qquad (1\leqslant i\leqslant n)$$

其中 a_1 称为基地址。图 2.1 给出了顺序表的存储结构。

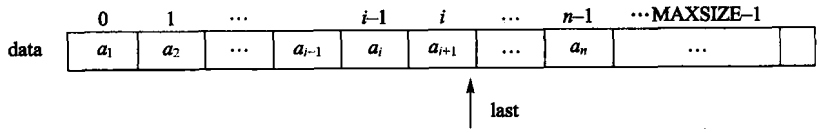


图 2.1 线性表的顺序存储示意图

顺序存储结构可以借助高级语言中的一维数组来实现，一维数组下标与元素在线性表中的序号相对应。用 C 语言描述如下：

```
#define MAXSIZE 1000          /*线性表能达到的最大长度*/
typedef  struct
  {   ElemType  data[MAXSIZE];
      int  last;
  } SeqList;
```

需要说明的是：元素的序号和数组的下标不是一致的，如 a_1 的序号是 1，在数组中的下标是 0。

2.2.2 顺序表基本运算的实现

（1）顺序表的初始化

顺序表的初始化即构造一个空表，将 L 设为指针参数，首先动态分配存储空间，然后，将表中 last 指针置为−1，表示表中没有数据元素。算法如下：

```
SeqList *init_SeqList( )
  { SeqList *L;
    L=(SeqList*)malloc(sizeof(SeqList));
    L->last=-1;
    return L;
  }
```

算法 2.1

（2）插入运算

线性表的插入是指在表的第 i 个位置上插入一个值为 x 的新元素，插入后使原表长为 n 的表：

$$(a_1,\ a_2,\ \cdots,\ a_{i-1},\ a_i,\ a_{i+1},\ \cdots,\ a_n)$$

成为表长为 $n+1$ 的表：

$$(a_1,\ a_2,\ \cdots,\ a_{i-1},\ x,\ a_i,\ a_{i+1},\ \cdots,\ a_n)$$

i 的取值范围为 $1 \leqslant i \leqslant n+1$。

顺序表上完成这一运算则通过以下步骤进行：

① 将 $a_i \sim a_n$ 顺序向后移动，为新元素让出位置；

② 将 x 置入空出的第 i 个位置；

③ 修改 last 指针（相当于修改表长），使之仍指向最后一个元素。

算法如下：

```
int  Insert_SeqList(SeqList *L, int i, datatype x)
 {  int j;
    if (L->last==MAXSIZE-1)
        {  printf("表满"); return(-1); }          /*表空间已满，不能插入*/
    if (i<1 || i>L->last+2)                      /*检查插入位置的正确性*/
        {  printf("插入位置错！"); return(0); }
    for(j=L->last;j>=i-1;j--)
        L->data[j+1]=L->data[j];                 /*向后移动元素 */
    L->data[i-1]=x;                              /*新元素插入*/
    L->last++;                                   /*last 仍指向最后元素*/
    return (1);                                  /*插入成功*/
  }
```

算法 2.2

在顺序表上做插入操作的时间复杂度为 $O(n)$。

（3）删除运算

线性表的删除运算是指将表中第 i 个元素从线性表中去掉，删除后使原表长为 n 的线性表：

$$(a_1,\ a_2,\ \cdots\ ,\ a_{i-1},\ a_i,\ a_{i+1},\ \cdots,\ a_n)$$

成为表长为 $n-1$ 的线性表：

$$(a_1,\ a_2,\ \cdots,\ a_{i-1},\ a_{i+1},\ \cdots,\ a_n)$$

i 的取值范围为：$1 \leqslant i \leqslant n$。

顺序表上完成这一运算的步骤如下：

① 将 $a_{i+1} \sim a_n$ 顺序向前移动；

② 修改 last 指针(相当于修改表长)使之仍指向最后一个元素。

算法如下：

```
int Delete_SeqList(SeqList *L;int i)
  { int  j;
    if(i<1 || i>L->last+1)                  /*检查空表及删除位置的合法性*/
        { printf ("不存在第 i 个元素"); return(0); }
    for(j=i;j<=L->last;j++)
        L->data[j-1]=L->data[j];            /*向前移动元素*/
    L->last--;
    return(1);                              /*删除成功*/
    }
```

算法 2.3

该算法的时间复杂度为 $O(n)$。

（4）按值查找

线性表中的按值查找是指在线性表中查找与给定值 x 相等的数据元素。在顺序表中完成该运算最简单的方法是：从第一个元素 a_1 起依次和 x 比较，直到找到一个与 x 相等的数据元素，则返回它在顺序表中的存储下标；或者查遍整个表都没有找到与 x 相等的元素，返回–1。

算法如下：

```
int Location_SeqList(SeqList *L, datatype x)
  {  int i=0;
     while(i<=L.last && L->data[i]!= x)
        i++;
     if (i>L->last)   return –1;
     else       return i;                    /*返回存储位置*/
}
```

算法 2.4

本算法时间复杂度为 $O(n)$。

2.2.3 顺序表应用举例

【例 2.1】 有顺序表 A 和 B，其元素均按从小到大的升序排列，编写一个算法将它们合并成一个顺序表 C，要求 C 的元素也是从小到大的升序排列。

算法思路：依次扫描 A 和 B 的元素，比较当前的元素的值，将较小值的元素赋给 C，如此直到一个线性表扫描完毕，然后将未完的另一个顺序表中余下部分赋给 C 即可。C 的容量要能够容纳 A、B 两个线性表相加的长度。

算法如下：

```
void merge(SeqList   A, SeqList   B,   SeqList *C)
{  int   i,j,k;
   i=0;j=0;k=0;
   while ( i<=A.last && j<=B.last )          /*归并*/
      if (A.data[i]<B.data[j])
            C->data[k++]=A.data[i++];
      else   C->data[k++]=B.data[j++];
   while (i<=A.last )                        /*插入 A 剩余部分元素*/
            C->data[k++]= A.data[i++];
   while (j<=B.last )                        /*插入 B 剩余部分元素*/
            C->data[k++]=B.data[j++];
   C->last=k–1;
}
```

算法 2.5

算法的时间性能是 $O(m+n)$，其中 m 是 A 的表长，n 是 B 的表长。

2.3 线性表的链式表示和实现

顺序表的存储特点是用物理上的相邻实现了逻辑上的相邻，它要求用连续的存储单元顺

序存储线性表中各元素，因此，对顺序表插入、删除时需要移动大量数据元素，大大影响了运行效率。本节介绍线性表链式存储结构，它不需要用地址连续的存储单元来实现，因为它不要求逻辑上相邻的两个数据元素物理上也相邻，它是通过“链”建立起数据元素之间的逻辑关系，因此对线性表的插入、删除不需要移动数据元素，从而提高了运行效率。

2.3.1 单链表

链表是将线性表的元素存储在一个链表的结点中，并用该结点中的指针指向线性表的下一个结点，链表中结点顺序与线性表中元素的逻辑顺序一致，但链表中相邻结点的存储位置不一定相邻。

为建立元素之间的线性关系，除了存放数据元素的自身信息（data）之外，还需要存放其后继元素存储位置的指针(next)，这两部分信息组成一个“结点”，结点的结构如图 2.2 所示。n 个结点的线性表通过每个结点的指针域拉成了一个“链子”，称为链表。又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。

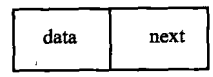


图 2.2 单链表结点结构

结点结构定义如下：

```
typedef struct
    {  datatype data;
       struct node *next;
   } LNode, *LinkList;
```

设有线性表$(a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8)$，图 2.3 给出了其链式存储结构，图 2.4 给出了其逻辑状态。将第一个结点的地址 160 放到一个指针变量如 H 中，最后一个结点没有后继，其指针域必须置空，表明链表到此结束，这样就可以从第一个结点的地址开始依次访问每一个结点。

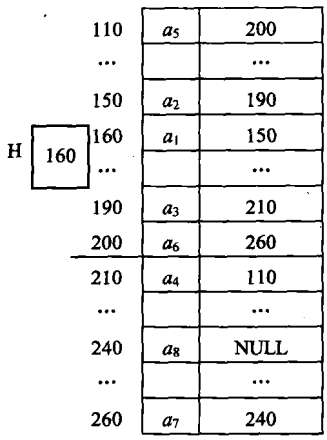


图 2.3 链式存储结构

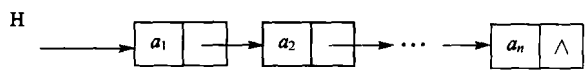


图 2.4 链表逻辑状态

通常用“头指针”来标识一个单链表，如单链表 H，是指某链表的第一个结点的地址放在了指针变量 H 中，头指针为“NULL”则表示一个空表。

2.3.2 单链表上基本运算的实现

（1）建立单链表

链表是一种动态管理的存储结构，链表中的每个结点占用的存储空间不是预先分配，而是运行时系统根据需求而申请的，因此建立单链表从空表开始，每读入一个数据元素则申请