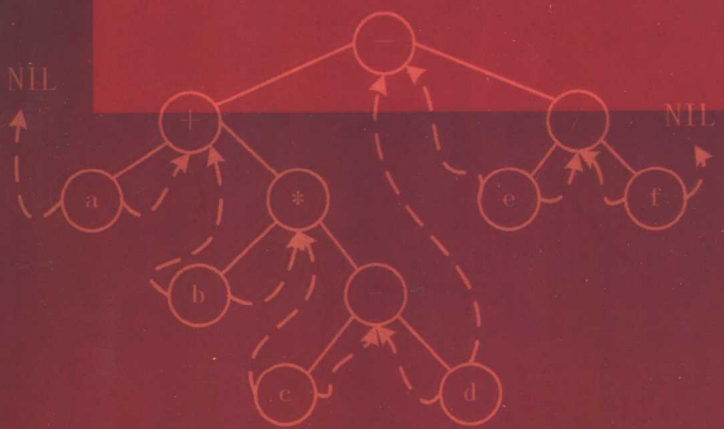
光盘附

数据结构经典算法实现与习题解答

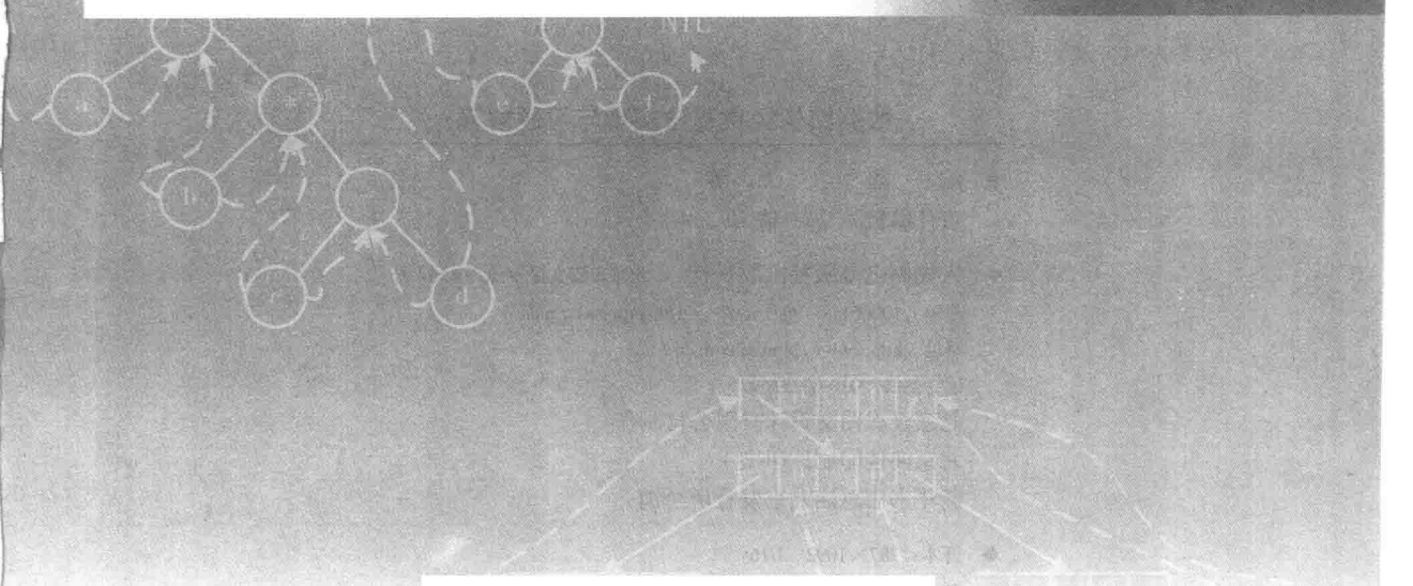
汪杰 等 编著



人民邮电出版社
POSTS & TELECOM PRESS

数据结构经典算法实现与习题解答

汪杰 等 编著



人民邮电出版社

图书在版编目（CIP）数据

数据结构经典算法实现与习题解答 / 汪杰等编著. —北京：人民邮电出版社，2004.1
ISBN 7-115-11753-5

I. 数… II. 汪… III. 数据结构—高等学校—自学参考资料 IV. TP311.12

中国版本图书馆 CIP 数据核字（2003）第 120408 号

内容提要

本书以计算机学科的主干课程——数据结构为主线索组织内容，涵盖了计算机学科涉及到的大部分经典算法的实现，书中结合大量的图示和程序代码，展示了各种数据结构的实现细节和编程技巧，能够帮助学习者提高运用知识解决实际问题的能力。同时，书中还汇编了高校数据结构常用教材中出现的大部分习题的解答。

本书的配套光盘中附带了书内提到的所有经典应用的 C 语言实现。

本书既可作为计算机及相关专业的学生学习数据结构、进行课程设计的辅导材料，又可作为学习者学习计算机学科其他相关课程的辅导材料。

数据结构经典算法实现与习题解答

◆ 编　　著　汪　杰　等
责任编辑　汤　倩

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www. ptpress.com.cn
读者热线　010-67132692
北京汉魂图文设计有限公司制作
北京鸿佳印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：787×1092　1/16
印张：21.75
字数：529 千字　　2004 年 1 月第 1 版
印数：1-5 000 册　　2004 年 1 月北京第 1 次印刷

ISBN 7-115-11753-5/TP・3674

定价：34.00 元（附光盘）

前　言

本书以计算机学科的主干课程——数据结构为主线索组织内容，涵盖了计算机学科的计算机算法设计、数值分析、操作系统、编译原理、离散数学等课程涉及的大部分经典算法的实现，同时，本书还汇编了高校数据结构常用教材中出现的大部分习题的解答。

1. 为什么写这本书

本书旨在帮助学习数据结构的广大读者掌握数据结构的原理和解题方法，吸取经典算法的设计思想及精髓，提高运用知识解决实际问题的能力，为今后的程序设计奠定扎实的基础。

2. 本书的特色

本书的重点在于使读者通过数据结构的思想，掌握程序设计中各种算法的实现细节及编程的技巧。

- **概念回顾**：对主要的数据结构概念，如数组、栈和队列、图和二叉树等，采用简明的语言进行了归纳总结。
- **数据结构描述及算法实现**：结合大量的图示和程序代码，展示了各种数据结构的实现细节和编程技巧。
- **经典应用**：给出大量经典问题的解题思路及算法实现，例如八皇后问题、汉诺塔问题、各种排序问题、迷宫问题和背包问题等。

本书汇聚了许多教材、参考书的精华，而且在光盘中附带了书内提到的所有经典应用的C语言实现。

3. 读者对象

本书既可作为计算机及相关专业的学生学习数据结构、进行课程设计的辅导材料，又可作为学习者学习计算机学科其他相关课程的辅导材料。

本书由汪杰主编，参与写作的人员有钟军、刘海英、白露、王占全、庞演、顾强等。

鉴于时间仓促，书中纰漏和考虑不周之处在所难免，热忱欢迎读者予以批评、指正，以便日后能够为您撰写出更好的图书。

读者在使用本书时，如果有什么问题、意见和建议，欢迎通过书后的读者信息反馈卡及时交流与沟通，或者发E-mail至tangqian@ptpress.com.cn，留下宝贵的意见。

编　者

2004年1月

光盘说明

为了方便读者学习，本书附带了一张光盘。下面对光盘内容及使用方法进行简要的介绍。

1．光盘的运行环境

硬件环境：CPU 的主频在 200MHz 以上、内存在 64MB 以上。

软件平台：操作系统为 Windows 98/Me/NT/2000/XP，调试环境为 Visual C++ 6.0 及以上版本或者 Turbo C 2.0 及以上版本。

2．光盘的使用方法及注意事项

将本书的源代码拷入硬盘中，用 Visual C++ 或者 Turbo C 打开文件，即可编译运行。

3．版权说明

本书附带的源码均是作者编写和测试过的，仅提供读者学习时使用，不能用作其他商业用途，否则责任自负。

4．光盘的主要内容

本书附盘中存放的是本书中涉及的源代码，包括基本算法描述、经典应用、常用教材习题解答。

5．光盘目录

内容	路径	内容	路径
转置矩阵	\03\3-2	在线等价类	\07\7-9
矩阵相乘	\03\3-3	直方图	\08\8-7
对角矩阵存储	\03\3-4	箱子装置问题	\08\8-8
三对角矩阵建立	\03\3-5	交叉分布	\08\8-9
三角矩阵建立	\03\3-6	寻找路径	\09\9-12
对称矩阵建立	\03\3-7	连通图	\09\9-13
串插入	\03\3-9	机器调度	\10\10-9
串匹配	\03\3-10	霍夫曼编码	\10\10-10
改进串匹配	\03\3-11	希尔排序	\11\11-2
串合并	\03\3-12	快速排序	\11\11-4
迷宫问题	\04\4-9	堆排序	\11\11-7
识别图元	\04\4-10	装船问题的贪心算法	\12\12-1
括号匹配	\04\4-11	拓扑排序的贪心算法	\12\12-2
火车车厢重排	\04\4-12	单源点最短路径贪心算法	\12\12-3
开关盒布线	\04\4-13	残缺棋盘的分治算法	\12\12-4
离线等价类	\04\4-14	选择问题的分治算法	\12\12-5
建立稀疏矩阵	\05\5-18	快速排序的分治算	\12\12-6
删除稀疏矩阵	\05\5-19	最短路径	\12\12-9
箱子排序	\05\5-21	旅行商问题的回溯算法	\12\12-10
汉诺塔	\06\6-2	装船问题的回溯算法	\12\12-11
八皇后问题	\06\6-3	电路板排列问题的回溯算法	\12\12-12
信号放大器设置	\07\7-8	货箱装载问题	\12\12-13

目　录

第 1 章　初识数据结构 1
1.1　数据结构回顾 1
1.1.1　数据结构的概念 1
1.1.2　系统的生命周期 2
1.2　算法的定义 3
1.3　算法性能分析和度量 7
1.3.1　算法评价的标准 7
1.3.2　算法复杂度的度量 7
1.4　数据抽象 12
1.5　常用教材习题解答 13
1.6　巩固练习 15
第 2 章　重要的 C 语言概念 17
2.1　基本数据类型回顾 17
2.1.1　整型数据 17
2.1.2　实型数据 17
2.1.3　字符型数据 18
2.2　基本控制语句 19
2.2.1　条件语句 19
2.2.2　循环语句 20
2.2.3　开关语句 27
2.3　结构和联合 30
2.3.1　结构 30
2.3.2　联合 35
第 3 章　数组 37
3.1　顺序数组 37
3.1.1　概念回顾 37
3.1.2　数据结构实现 37
3.2　实例－多项式的数组表示 38
3.3　实例－稀疏矩阵 39
3.3.1　概念回顾 39
3.3.2　矩阵置换 40
3.3.3　矩阵乘法 42
3.3.4　对角矩阵建立 44
3.3.5　三对角矩阵建立 45
3.3.6　三角矩阵建立 46
3.3.7　对称矩阵建立 47
3.4　字符串数组 48
3.4.1　基本操作实现 48
3.4.2　实例－串插入、串匹配及改进、串合并 49
3.5　常用教材习题解答 51
第 4 章　栈和队列 57
4.1　栈 57
4.1.1　概念回顾 57
4.1.2　数据结构实现 57
4.1.3　基本操作实现 58
4.1.4　实例－进制转换问题 62
4.2　队列 64
4.2.1　概念回顾 64
4.2.2　数据结构实现 64
4.2.3　基本操作实现 65
4.2.4　实例－舞伴问题 71
4.3　经典应用 74
4.3.1　迷宫问题 74
4.3.2　图元识别 76
4.3.3　括号匹配 79
4.3.4　火车车厢重排 81
4.3.5　开关盒布线 85
4.3.6　离线等价类 88
4.4　常用教材习题解答 91
第 5 章　链表 97
5.1　单链表 97
5.1.1　概念回顾 97
5.1.2　数据结构实现 97
5.1.3　基本操作实现 98
5.1.4　实例－动态堆栈、动态队列 108
5.2　双向链表 113
5.2.1　概念回顾 113
5.2.2　数据结构实现 113
5.2.3　基本操作实现 113
5.3　循环链表 116
5.3.1　概念回顾 116
5.3.2　数据结构实现 116
5.3.3　基本操作实现 116
5.4　实例－循环链表的多项式求解 117
5.5　实例－稀疏矩阵 121
5.5.1　稀疏矩阵的建立 123

5.5.2 稀疏矩阵的删除 125
5.6 经典应用—箱子排序 128
5.7 常用教材习题解答 130
第 6 章 递归 139
6.1 概念回顾 139
6.2 实例—汉诺塔 140
6.2.1 算法实现 140
6.2.2 递归过程分析 141
6.3 经典应用—八皇后问题 145
6.4 常用教材习题解答 147
第 7 章 树和森林 150
7.1 树和森林 150
7.1.1 概念回顾 150
7.1.2 数据结构实现 151
7.2 二叉树 154
7.2.1 概念回顾 154
7.2.2 数据结构实现 156
7.3 二叉树的遍历 159
7.3.1 中序遍历 159
7.3.2 前序遍历 160
7.3.3 后序遍历 161
7.3.4 应用实例 162
7.4 线索化二叉树 162
7.5 树与森林的转换 165
7.5.1 森林与二叉树的转换 . 165
7.5.2 森林的遍历 166
7.6 经典应用 168
7.6.1 信号放大器 168
7.6.2 在线等价类 171
7.7 常用教材习题解答 173
第 8 章 搜索 190
8.1 静态搜索 190
8.1.1 概念回顾 190
8.1.2 实例—顺序搜索 191
8.1.3 实例—基于有序表的折半搜索 192
8.2 二叉搜索树 194
8.2.1 概念回顾 194
8.2.2 基本操作实现 196
8.3 AVL 树 201
8.3.1 概念回顾 201
8.3.2 基本操作实现 202
8.4 经典应用 205
8.4.1 直方图 205
8.4.2 箱子装载问题 206
8.4.3 交叉分布 208
8.5 常用教材习题解答 212
第 9 章 图 218
9.1 概念回顾 218
9.2 数据结构实现 218
9.2.1 图的邻接矩阵表示法 . 218
9.2.2 图的邻接表表示法 220
9.3 基本操作实现 223
9.3.1 深度优先搜索 223
9.3.2 广度优先搜索 225
9.3.3 连通分量 228
9.3.4 生成树 230
9.3.5 重连通分量 231
9.4 图的最小生成树 234
9.4.1 Kruskal 算法 235
9.4.2 Prim 算法 238
9.5 求解最短路径 240
9.5.1 单源全目的地的最短路径 241
9.5.2 所有对的最短路径 243
9.6 实例—活动网络（AOV） .. 244
9.7 经典应用 246
9.7.1 寻找路径 246
9.7.2 连通图 247
9.8 常用教材习题解答 249
第 10 章 堆和哈希表 257
10.1 堆 257
10.1.1 概念回顾 257
10.1.2 基本操作实现 257
10.2 哈希表 262
10.2.1 概念回顾 262
10.2.2 基本操作实现 263
10.2.3 静态哈希 267
10.2.4 动态哈希 270
10.3 经典应用 270
10.3.1 机器调度 270
10.3.2 霍夫曼编码 272
10.4 常用教材习题解答 274
第 11 章 排序问题 276
11.1 概念回顾 276

11.2 插入排序 276
11.3 交换排序 280
11.3.1 冒泡排序 280
11.3.2 快速排序 283
11.4 归并排序 285
11.4.1 概念回顾 285
11.4.2 自底向上的实现 285
11.4.3 自顶向下的实现 287
11.5 堆排序 289
11.5.1 概念回顾 289
11.5.2 数据结构实现 289
11.6 基数排序 290
11.6.1 概念回顾 290
11.6.2 链式排序实现 293
11.7 外排序 295
11.7.1 概念回顾 295
11.7.2 K 路平衡归并 295
11.8 常用教材习题解答 297
第 12 章 经典算法设计 307
12.1 贪心算法 307
12.1.1 算法思想及描述 307
12.1.2 经典应用 309
12.2 分治算法 315
12.2.1 算法思想及描述 315
12.2.2 经典应用 316
12.3 动态规划 322
12.3.1 算法思想及描述 322
12.3.2 经典应用 322
12.4 回溯算法 326
12.4.1 算法思想及描述 326
12.4.2 经典应用 328
12.5 分枝定界算法 336
12.5.1 算法思想及描述 336
12.5.2 经典应用 337

第 1 章　初识数据结构

随着社会的发展，计算机已经成为了人类不可或缺的工具和帮手，在社会生活中发挥越来越大的作用。如今，计算机已经深入到社会的各个领域，我们再也无法想象没有计算机的生活会是怎样。银行信用卡、IC 电话卡、电子户籍、股票账户等都成了日常生活中必不可少的一部分，而计算机也已经突破了原来单纯的科学计算功能，成为人们存储、处理数据的重要途径。如何高效地存储、查找、分析数据，特别是对海量数据的处理，成为程序设计人员非常关心的一个话题。“数据结构”这门学科正是在这样的背景下成长和发展起来的。

1.1　数据结构回顾

1.1.1　数据结构的概念

大千世界，变化万千，信息无时不在，无处不在。如前面提到的银行信用卡里的金额是信息，IC 电话卡里的金额是信息，股票的价格是信息，你的户籍也是信息，信息以各种各样不同的形式和方式存在于每个人的周围。但是纷繁芜杂的信息是无法被计算机识别和处理的，只有当信息被转换成为数据的时候才能被计算机管理。据此可以认为：数据【Data】是对客观事物的符号表示，在计算机科学中是指所有能输入到计算机中并被处理的符号的总称。

根据一般的观点，数据可以被分为两大类：一类是整数、实数、复数等数值性数据；另一类是图像、声音、文字等非数值性数据。随着多媒体和图形学的发展，越来越多的非数值性数据正在被人们所使用。

数据元素【Data Element】是数据的基本单位，在计算机程序中通常作为一个整体存在，被计算机加工和处理。但有时数据元素也是可分的。例如，在电子户籍档案中，人是作为数据元素存在的，但是他还可以被分成姓名、年龄、性别等许多项目。

在计算机应用中，人们总是把类似性质的数据元素归类到一些集合中去，这些集合被称之为数据对象【Data Object】。例如，整数数据对象的集合为 $\mathbf{Z}=\{0,\pm1,\pm2,\cdots\}$。在了解了数据和数据对象以后，可以给数据结构下一个定义。

数据结构【Data Structure】是相互之间存在一种或多种特定关系的数据元素的集合。在现实问题中，数据元素之间都是相互联系的。任何一个数据元素可能和一个或者多个其他的数据元素有着这样或者那样的关系，这种关系称为结构【Structure】。这些关系概括起来有集合关系、一对一、一对多、多对多 4 种关系。集合关系指的是两个数据元素存在于同一个数据集合；一对一指的是结构中的元素是一个对一个的关系；一对多是指结构中的元素可以是一个对多个，形成树状，如图 1-1-1(a)所示；而多对多是指数据元素存在多个对多个的关系，形成网状，如图 1-1-1(b)所示。

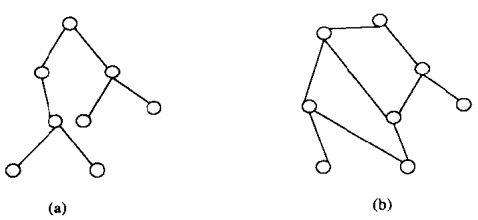


图 1-1-1　数据结构图

最后讨论数据存储的问题。所谓数据存储就是数据结构在计算机内存中的表示。表示要包括两个方面的内容：

- 数据元素；
- 数据元素的逻辑关系。

这两个方面正好与定义数据结构是相互之间存在一种或多种特定关系的数据元素的集合相对应。通常基本的存储结构有以下两种。

- 顺序存储结构：在一组连续的存储单元中存放数据，用数据元素在内存空间中的相对位置来表示某种特定的关系。例如，C 语言中的数组结构。
- 链式存储结构：用一组任意的存储单元存放数据元素，通过保存相关数据元素的存储位置来表示数据之间特定的关系。例如，C 语言中的链表结构。

1.1.2 系统的生命周期

在学习数据结构之前，先来了解一下系统的生命周期。对于一个好的程序员来说，他会将一个大的计算机程序看成是许多相互影响、复杂的系统。作为一个系统，程序员将经历一个完整的程序开发过程，称之为系统的生命周期。这一周期包括需求、分析、设计、编程以及调试等一系列阶段。

1. 需求

所有的大工程都是从对工程目的的描述开始的。这些描述包含了程序员所获得的全部原始信息以及用户对程序的期望。这些信息在一开始都是比较模糊的，这就要求程序员设计包含系统所有输入输出情况的严格描述。

2. 分析

在详细描述了系统的需求以后，就进入了分析阶段。通常程序员会采用自顶向下和自底向上两种分析方法分析程序。对于这两种分析方法的具体描述涉及到软件工程的知识，读者如果有兴趣可以参阅其他软件工程的书籍。在这一阶段主要是设计和筛选工程中遇到的问题的解决方案。

3. 设计

这一阶段是分析阶段的继续。设计者通过描述在程序中用到的数据对象以及对这些对象的操作来实现系统功能。对数据对象的描述导致了建立起一些抽象的数据结构，而对数据对象操作则涉及到对算法的描述以及对算法的比较策略。例如，如果要为某大学设计一个课程安排管理系统，典型的数据对象一般包括学生、课程和教授，典型的操作一般包括插入、删除以及查询某几个数据对象。由于抽象数据类型和算法是用语言描述的，所以在这一步不考虑代码实现的细节，但是必须说明每个数据对象的基本信息。例如可以说明学生数据对象需要包括姓名、学号、专业和电话号码等。

4. 编码

在这个阶段，要求选择对数据对象的具体实现方式并且为每种操作写出算法。这是非常关键的一步，因为对数据对象的具体实现方式直接决定了与之相关的算法的效率。这说明，一般算法都是独立与数据对象的。

5. 调试

这一阶段包括对程序的正确性的证明，用各种不同的输入数据来测试程序，以及对错误的修改。这是 3 个比较复杂的方面，对此的研究也是比较多的，读者可以参考软件工程方面的书

来了解这些内容。

1.2　算法的定义

上一节中介绍了算法很多时候是独立于数据对象存在的，也就是说算法有很大的独立性。这一节主要讨论有关算法的问题。

一个程序应包括两个重要的部分。

- 对数据的描述：在程序中要指定数据的类型和数据的组织形式，即数据结构。
- 对操作的描述：即操作步骤，也就是算法【Algorithm】。

计算机的著名学者 Nikiklaus Wirth 提出了这样的公式：

数据结构+算法=程序

可见算法对于程序来说是多么重要。那么首先来回答一个问题：什么是算法。

算法【Algorithm】一词是由算术【Algorism】衍生而来的。一般字典的解释是“解决一类确定问题的任何一种特殊方法”。在计算机科学中算法指的是用计算机解决一个问题的精确而有效的方法。这是一个在计算机科学中的基本概念。算法是针对于许多普遍问题的，一个好的算法对于实现一个高效的程序有着非常巨大的意义。可以给算法如下的定义。

算法：一个算法是一个能被机械执行的动作的有穷集合，一个动作的一次执行称为一步。

同时作为算法必须包含以下 5 个特征。

输入：一个算法有 0 个或多个输入量。这些输入量是算法要求的初始信息，它们取自某一个特定的集合。

输出：一个算法有一个或多个输出信息。它们通常是同输入信息有特定联系的量。

确定性：算法的每一步都必须有确定的意义，动作不能有二义性。例如，不允许出现“计算 7/0”这样的无意义的步骤和“将 3 或者 5 与 4 相加”这样的有二义性的算法。

有穷性：一个算法对任何一个合法输入必须在执行有限次步骤以后终止。

有效性：也叫可行性，指算法的每一步动作必须是相当基本的，也就是说，每一步至少在原理上是可以由人在有限的时间内用笔和纸来完成的。

在计算的理论中，关于算法和程序的一个重要的区别是，程序不满足算法的第 4 个条件，即有穷性。例如，操作系统中的等待循环，系统会一直等待，直到有任务唤醒系统去执行任务。可见，系统是不会停止运行的，除非系统崩溃了，所以这是一个程序而不是算法。

可以用很多方法描述一个算法。可以用自然语言像英语、中文、日文等；如果用了自然语言，就必须保证结束动作是明确的。图描述也是一个不错的选择。概括起来有以下几种表示算法的方式：

（1）用自然语言表示算法，但除了很简单的问题，一般不用自然语言表示算法；

（2）用流程图表示算法，流程图表示算法，直观形象，易于理解；

（3）用 N-S 流程图表示算法，这是 1973 年美国学者提出的一种新型流程图；

（4）用伪代码表示算法，伪代码使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。

我们的任务是用计算机解题，就是用计算机实现算法，用计算机语言表示算法必须严格遵循所用语言的语法规则。

在本书中，使用的描述是用 C 语言来实现的，有时为了便于理解，也用 C 语言和英语的

混合语言来实现。但读者学习中必须明确，算法可以用任何语言描述。接下来先通过两个简单的算法来介绍一下算法的表述。

【例 1.1】求 1×2×3×4×5=?

这个算法非常简单，就是将各个数相乘，下面给出算法的自然语言描述。

【程序 1-1】计算 1×2×3×4×5

步骤 1：先求 1×2，得到结果 2。

步骤 2：将步骤 1 得到的乘积 2 乘以 3，得到结果 6。

步骤 3：将 6 再乘以 4，得 24。

步骤 4：将 24 再乘以 5，得 120。

这是一个比较简单的算法，可以用自然语言比较清楚地描述。但如果算法比较复杂，那么自然语言就不是好的方法了。下面的这个算法就比较复杂，所以不易直接用自然语言来描述。

【例 1.2】选择排序问题：设要将一组含有 $n(n>1)$个整数的集合进行排序。一个简单的方法：从未排序的集合中取出最小的一个整数放到另一个已经按从小到大排好序的整数队列中。

虽然上面的陈述已经足够地描述了问题的解决办法，但这仍然不是一个算法，因为还有许多没有描述清楚的问题。例如，它没有解释原来的整数集合放在哪里，如何放置，新的整数队列放在哪里。

假设整数是以整型数组 list 的形式存储的，第 i 个整数存放的位置为 list[i]($0<i<n$)。程序 1-2 是一个用 C 语言和中文注释给出的解决方法。

【程序 1-2】选择排序算法

```
for ( i = 0; i < n; i++ )
{
    //遍历 list[i]到 list[n-1],找出最小的整数 list[min];
    //交换 list[i]和 list[min]的值;
}
```

要将程序 1-2 转换成真正的 C 语言代码有两个问题需要解决：一是找到最小整数 list[min]，二是交换 list[i]和 list[min]。程序 1-3 给出了整个程序的 C 语言实现。

【程序 1-3】选择排序

```
#include <stdio.h>
#include <math.h>
#define     Max_Size 101
#define SWAP(x,y,t) ((t) = (x), (x) = (y), (y) = (t))
void sort(int [], int);
void main(void)
{
    int i, n;
    int list[Max_Size];
    printf("输入整数的个数:");
    scanf("%d",&n);
    if( n < 1 || n > Max_Size)
    {
        printf("整数太多,错误!");
        exit(1);
    }
```

```
    for ( i = 0; i < n; i++ )
    {
        list[i] = rand() % 1000;
        printf("%d    ",list[i] );
    }
    sort(list,n);
    printf("\n 排列好的数组是:\n");
    for ( i = 0; i < n; i++ )
        printf("%d    ",list[i]);
}
void sort(int list[], int n)
{
    int i, j, min, temp;
    for ( i = 0; i < n-1; i++)
    {
        min = i;
        for ( j = i + 1; j < n; j++)
            if ( list[j] < list[min]) min = j;
            SWAP( list[i], list[min], temp);
    }
}
```

在程序中，对第一个问题，是通过调用子函数 sort 来实现的。在 sort 函数中，假设最小的整数是 list[i]，通过比较 list[i],list[i+1],list[i+2],…,list[n-1]，发现比 list[i]小的数字时就把该数设为最小数。接着就使用宏定义 SWAP 来实现数值的交换。

对第二个问题，即数据的交换，在第 4 行使用了一个宏定义方便地解决了。当然读者还可以使用函数调用的方法来解决这个问题。程序 1-4 给出了用函数实现交换的方法，有兴趣的读者可以用程序 1-4 的函数来改写程序 1-3，看看是否可以实现。

【程序 1-4】交换函数

```
void swap( int* x, int* y)
{
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

注意：交换函数的参数必须是指针，这一点读者要引起充分的重视。在以后的学习中，读者要注意体会每个程序中指针使用的场合和意义。

【例 1.3】二叉搜索：假如有 $n \geq 1$ 个不同的整数，它们已经被排好顺序放在一个数组 list 里。也就是说，list[0]<list[1]<list[2]…<list[n-1]。设计一个程序，当要找的数字 sn 在数组里时，返回数字所在的位置，否则返回-1 表示要找的数字 sn 不在数组里。

使用的算法描述如下：设有 left 和 right 两个变量分别表示要被搜索数组的最左端和最右端。开始的时，left 为 0，right 为 n-1。设 middle=(left+right)/2 表示数组的中间位置。对要搜索的数字 sn 和数组中间位置的值 list[middle]进行比较可以有以下 3 种可能的结果。

（1）sn<list[middle]，这种情况表示假如 sn 存在于当前的数组中，那么它必存在于 left 和 midlle-1 之间。此时把 right 的值设为 middle-1。

（2）sn>list[middle]，这种情况表示假如 sn 存在于当前的数组中，那么它必存在于 right 和 midlle+1 之间。此时把 left 的值设为 middle+1。

（3）sn=list[middle]，这种情况表示找到 sn，那么将中间位置值 middle 返回即可。

程序反复执行比较、赋值这两个操作，直到 sn 被找到或者数组中所有的数据都被比较，但没有找到 sn 了才结束。程序 1-5 给出了这个算法。

【程序 1-5】二叉搜索

```
while (数组中还存在着未被处理的数据)
{
    middle = ( left + right ) / 2;
    if (sn < list[middle])
        right = middle -1;
    else if(sn > list[middle])
        left = middle + 1;
    else return middle;
}
```

可以用两种方法实现比较，和程序 1-3 一样，一是用宏定义，二是用函数实现。当两个数相比较时通过返回值来判断两个数的大小关系。例如，比较两个数 A 和 B：

（1）如果 A < B，返回-1；

（2）如果 A > B，返回 1；

（3）如果 A = B，返回 0。

程序 1-6 是用函数表示的比较方法。宏定义比较简单，可以表示成：

```
#define COMPARE(X , Y)    (((X)<(Y))?－1：((X)==(Y))?0 ： 1)
```

【程序 1-6】比较两个整数

```
int   compare( int x, int y)
{
    if ( x < y)   return -1;
    else if ( x > y)   return 1;
    else return 0;
}
```

而程序 1-7 列出了使用了比较函数以后，比较完整的二叉搜索算法的 C 语言描述。函数 binsearch 将在 list 数组里面搜索 sn。如果找到则返回它的位置，否则返回-1 表示没有找到。

【程序 1-7】二叉搜索算法

```
int binsearch( int list[], int sn, int left, int right)
{
    int middle;
    while (left <= right)
    {
        middle = (left + right)/2;
        switch (compare(list[middle], sn))
        {
        case -1: left = middle +1;
                 break;
        case 0 : return middle;
        case 1 : right = middle - 1;
        }
    }
    return -1;
}
```

1.3　算法性能分析和度量

在学习了什么是算法以后就涉及到对算法进行评价和筛选的问题，因为在处理某个特定的问题时往往可以设计出许多不同的算法。而算法的好坏对能否解决问题有着极其重要的意义。以破解密码的算法为例，从理论上来说，通过穷举法列举所有可能的输入字符的组合情况可以破解任何的密码，但是如果密码比较长，那么破解工作就有可能需要很长的时间，可以长达几年、十几年甚至几百年。这样的算法显然是没有什么实际意义。有些好的算法可能将时间缩短到几天或者几个小时，那么这样的算法就比较有实际应用价值了。一般情况下，人们总是希望以较少的代价来完成较多的工作，所以设计和应用一个好的算法对于程序设计人员来说是至关重要的。

1.3.1　算法评价的标准

怎么才算一个好的算法呢？这就涉及到对算法性能标准的评价问题了。根据实际情况的不同，对算法的评价标准有很多，但以下的评价标准是其中最主要的一些。

- 正确性：这是算法最基本也是必不可少的一个标准。所谓正确性就是要求算法可以正确地执行预先规定的功能和性能要求。只有正确无误地完成了预先设计要求的算法才可以评价算法的优和劣。
- 可用性：所谓可用性就是要求算法能方便地被使用。一个好的算法要求其能被用户容易地使用，并能通过简单修改应用于同类问题地解决。一般算法设计人员总是通过模块化的设计、简单而有效的参数传递来实现算法的可用性。
- 可读性：与可用性相类似，算法的可读性也是为了实现算法与实际问题的相对独立性和通用性。这就要求算法可以为其他使用者容易地理解和修改。
- 高效性：算法的高效性就是要求算法有比较好的执行效率。在计算机中算法的执行都需要消耗计算机的资源，包括算法运行对系统时间的占用以及对存储空间的占用。前者被称做算法的时间代价，后者被称做算法的空间代价。两者都是数据结构这门课程中主要研究的问题。
- 容错性：有些书也称其为健壮性，是指算法对于某些错误的输入必须能够自动检测、报错，如果有可能还可以对错误进行一定程度上的纠正。这一标准是为了保证程序在出现一些小的错误的情况下仍然可以运行，而不会因为用户的一点点失误而导致系统问题。

以上虽然是算法评价的一般标准，但是出于不同的目的，对于不同的应用可能会有一些特殊的算法评价标准。

1.3.2　算法复杂度的度量

在上一小节提到，算法的一个重要评价标准是高效性，其中包括对算法时间和空间代价的度量。但是对于不同的计算机、不同的程序系统，这两个参数是很难比较的。例如，用 Unicode 表示的字符型的数据所占据的空间是 16bit，而用 ASCII 编码表示时所占的空间就 6bit。再如，用一台奔腾 4 处理器的计算机来运行某一个算法通常会比在一台老式的奔腾 2 上运行的时间要短一些。先来了解一下有关算法的几个概念。

1. 问题的规模【Size of Problem】

输入量大小的一个度量，又称为问题的大小，一般用 n 表示。在不同的实际问题中有不同

的含义。例如，对于图像来说，n 可以是边数或者顶点个数；对于破译密码的算法来说，n 可以是密码的长度也可以是密码每一位可以采取的待选字符个数。

2. 空间复杂度【Space Complexity】

依据算法编出程序以后，在计算机中占用的存储单元的总数，包括程序本身的长度和存储工作单元的长度，一般用 $S(n)$表示。在处理数据量较大的算法中，程序本身所占的空间一般是比较小的。不难看出 $S(n)$是随着 n 的增长而增长的。其极限情况称为渐近空间复杂度。

3. 时间复杂度【Time Complexity】

依据算法编出程序以后，运行程序在计算机中消耗的时间，一般用 $T(n)$表示。显然 $T(n)$也是随着 n 的增长而增长的。其极限情况称为渐近时间复杂度。

4. 空间复杂度的计算

一般说，程序的空间代价与特定计算机中数据的存放形式有关系。存储形式不同，对空间复杂度的影响也不同。可以用一个规定来解决。假设算法中所讨论的基本类型数据，不论是整数、字符，也不论是用什么编码形式表示，均可以存放在一个单元中，分析算法空间复杂度的时候，不考虑计算机单元的大小。有了这个规定以后，就可以计算一个程序 P 所需的空间 $S(P)$，它由以下两个部分组成。

- 确定的空间：这部分空间指的是不依赖于程序输入的数据的大小，也就是不依赖于问题规模的空间。对于一个特定的程序，这是一个特定的值。通常包括代码指令的大小，简单变量，固定大小的结构变量（例如 C 语言中的 struct）和常量。一般用常数 C 表示其大小。
- 变化的空间：这部分空间包括变化大小的结构变量，通常这些结构的大小由具体的实际问题决定，受输入规模 n 的影响。如果是递归程序，还包括额外的空间开销。一般用 $S_P(n)$表示某个程序 P 对于特定输入规模 n 的函数。

所以某一个程序所需的空间表示为：

$$S(P)=C+S_P(n)$$

由于 C 通常是个常数，所以在空间复杂度的计算当中，一般只计算 $S_P(n)$来表示特定算法的空间复杂度。如果某个算法没有变化的空间需求，则算法的空间复杂度为 0。

5. 时间复杂度的计算

程序运行的时间与下列因素有关：

- 程序的输入；
- 编译的目标代码的质量；
- 执行程序机器指令的性质和速度；
- 构成程序的算法的时间复杂度。

正因为有如此多的因素，为了能比较客观的评价和比较算法，有必要分析一下各种因素对算法的影响以及如何正确处理这些因素。

运行时间是输入规模的函数 $f(n)$，但是要记住 $f(n)$不等同于要求的时间复杂度 $T(n)$。由于在实际情况中，程序的输入不是一个确定的值 n，而是一个不确定的输入量，n 值是表示输入数据的规模。这就涉及到两个重要的概念：最坏时间复杂度和平均时间复杂度。

最坏时间复杂度：规模为 n 的所有输入量程序运行时间的最大值。

平均时间复杂度：规模为 n 的所有输入量程序运行时间的平均值。

由于平均时间复杂度比最坏时间复杂度要复杂，所以常常通过求最坏时间复杂度来表示某个算法的时间复杂度。但是，必须要记住这两者是有区别的，也就是说，最坏时间复杂度最小的算法不一定是平均时间复杂度最小的算法。

由于算法的执行时间和运行程序的计算机有着密切的关系，所以 $T(n)$不能直接表达成 n 的函数，而要用“阶”来表示。

- O-$f(n)$ = $O(g(n))$：表示当且仅当存在两个正常数 C 和 n_0，使得对所有的 $n \geqslant n_0$，有 $|f(n)| \leqslant C|g(n)|$；
- Ω-$f(n)$ = $\Omega(g(n))$：表示当且仅当存在两个正常数 C 和 n_0，使得对所有的 $n \geqslant n_0$，有 $|f(n)| \geqslant C|g(n)|$；
- θ-$f(n)$ = $\theta(g(n))$：表示当且仅当存在两个正常数 C_1,C_2 和 n_0，使得对所有的 $n \geqslant n_0$，有 $C_1|g(n)| \leqslant |f(n)| \leqslant C_2|g(n)|$。

容易得到以下的一些重要结论：

- 多项式 $A(n)=a_m n^m+a_{m-1}n^{m-1}+\cdots a_1 n+a_0 = O(n^m)$；
- $n(n+1)/2=\theta(n^2)$；
- $1+i^1+i^2\cdots i^n=O(n^{k+1})=\theta(n^{k+1})$；
- 若 $T_1(n)$和 $T_2(n)$分别是 P_1 和 P_2，它们的 $T_1(n)=O(f(n))$，$T_2(n)=O(g(n))$，则 $T_1(n)+T_2(n)=O(\max(f(n),g(n))$；$T_1(n)\times T_2(n)=O(f(n)\times g(n))$。

在 C 语言表示的算法中，算法的时间复杂度一般与程序执行的步骤数有关系，一般是所有步骤的时间复杂度相加。所以需要对一些语句的运行时间做出估计。

- 算术运算时间为 $O(1)$；
- 逻辑预算时间为 $O(1)$；
- 赋值运算时间为 $O(1)$；
- if 语句的运行时间为测试语句运行时间与后续执行语句运行时间的和；
- while 语句的运行时间为每次执行循环体的时间与循环次数之积；
- for 语句的运行时间与 while 相似；
- return 语句的运行时间为 $O(1)$；

由于存在重要结论 $T_1(n)+T_2(n)=O(\max(f(n),g(n))$，所以一个程序的时间复杂度是由程序中最复杂的部分组成。这一点是非常有用的。

接下来通过分析一些算法的实例来了解如何运用这些理论。

【例 1.4】求长度为 n 的整数数组 A 中的最大值。程序 1-8 描述了其算法，求其时间复杂度。

【程序 1-8】求数组中的最大数

```
int max(int A[], int n)
{
    int Max, i;                          //语句 3
    Max = A[0];                          //语句 4
    for (i = 1; i <= n-1; i++)           //语句 5
        if (A[i] > Max)                  //语句 6
            Max = A[i];                  //语句 7
    return Max;
}
```