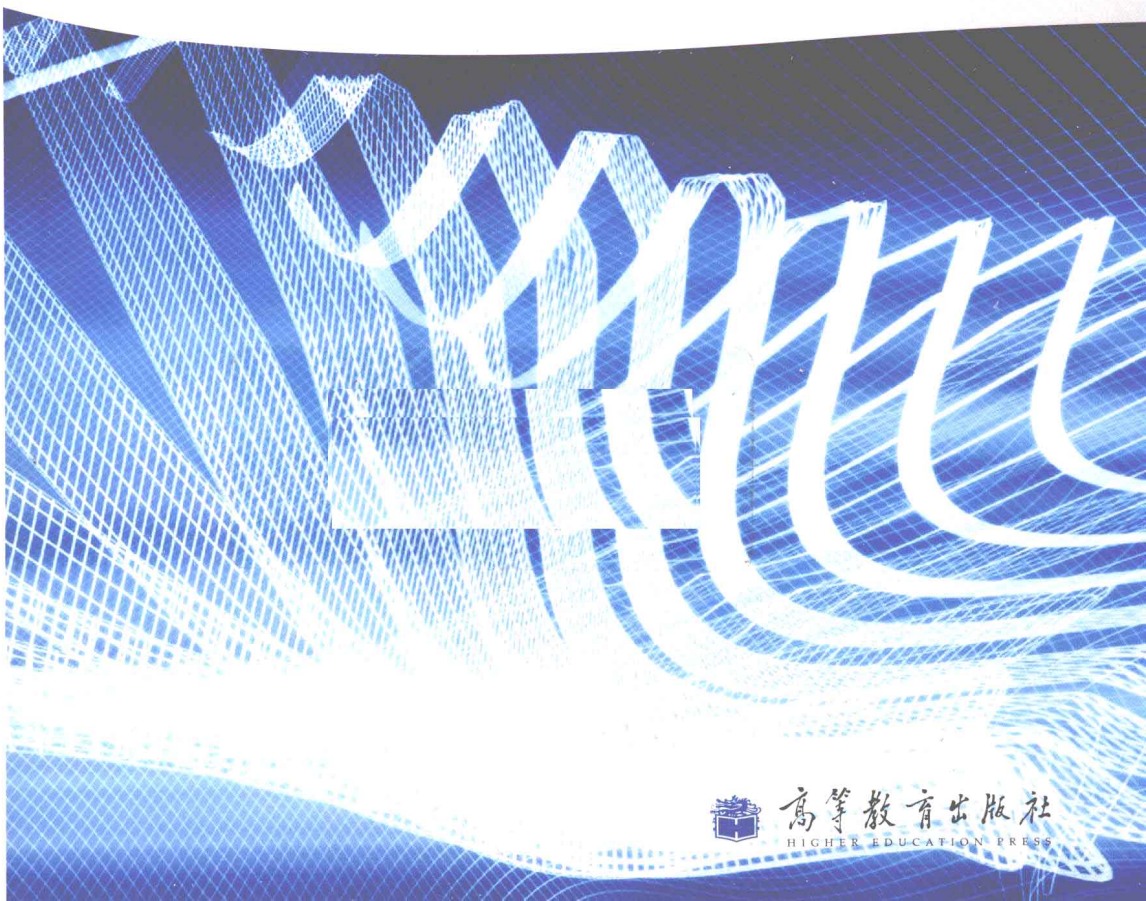
教育部高等理工教育教学改革与实践项目研究成果

《高等学校计算机科学与技术专业核心课程教学实施方案》规划教材

数据结构与算法

Data Structures and Algorithms

陈卫卫　王庆瑞　编著



高等教育出版社
HIGHER EDUCATION PRESS

教育部高等理工教育教学改革与实践项目研究成果

《高等学校计算机科学与技术专业核心课程教学实施方案》规划教材

数据结构与算法

Shuju Jiegou yu Suanfa

陈卫卫　王庆瑞　编　著

高等教育出版社·北京
HIGHER EDUCATION PRESS　BEIJING

内容简介

本书依据《高等学校计算机科学与技术专业核心课程教学实施方案》，面向计算机专业应用型人才培养的要求编写，内容包括顺序表、链表、栈、队、矩阵、字符串、广义表、树、二叉树、检索树、哈夫曼树、判定树、散列表，最优检索树、AVL树、红黑树、B树、B+树、2-3树、Trie树、union-find树、图等结构，各结构的特点和存储方法，实现查找、插入、删除、遍历、搜索的算法设计方法和时空效率；图的最小生成树和最短路径求解算法、内排序算法、文件结构和外排序算法；问题的固有难度、算法设计的一般方法、数据结构的类封装方法等；并配有400多道习题及部分习题的参考答案。

本书语言通俗流畅，叙述简洁，可读性强，并配有完整的PPT课件（可免费下载），可作为普通高校、职业学校、远程教育的教材和教学参考书，或程序设计爱好者的理论指导书。

图书在版编目 (CIP) 数据

数据结构与算法 / 陈卫卫，王庆瑞编著. —北京：高等教育出版社，2010.11
ISBN 978-7-04-031047-4

Ⅰ. ①数… Ⅱ. ①陈… ②王… Ⅲ. ①数据结构－高等学校－教材 ②算法分析－高等学校－教材
Ⅳ. ①TP311.12

中国版本图书馆 CIP 数据核字（2010）第 184837 号

策划编辑 倪文慧　**责任编辑** 郭福生　**封面设计** 张志奇　**责任绘图** 尹　莉
版式设计 马敬茹　**责任校对** 杨凤玲　**责任印制** 朱学忠

出版发行	高等教育出版社	**购书热线**	010-58581118
社　　址	北京市西城区德外大街 4 号	**咨询电话**	400-810-0598
邮政编码	100120	**网　　址**	http://www.hep.edu.cn
			http://www.hep.com.cn
		网上订购	http://www.landraco.com
经　　销	蓝色畅想图书发行有限公司		http://www.landraco.com.cn
印　　刷	北京铭传印刷有限公司	**畅想教育**	http://www.widedu.com
开　　本	787 x 1092　1/16	**版　　次**	2010 年 11 月第 1 版
印　　张	26.5	**印　　次**	2010 年 11 月第 1 次印刷
字　　数	590 000	**定　　价**	38.40 元

物料号　31047-00

前 言

为了更好地贯彻教育部高等学校计算机科学与技术教学指导委员会发布的《高等学校计算机科学与技术专业发展战略研究报告暨专业规范（试行）》和《高等学校计算机科学与技术专业公共核心知识体系与课程》的指导精神，加速建设优质教学资源，落实分类培养的战略目标，解决相关课程建设中存在的问题，教育部高等学校计算机科学与技术专业教学指导分委员会、中国计算机学会教育专业委员会、高等教育出版社相联合，从全国30多所高校中遴选富有教学经验的教师和专家，就“高等学校计算机科学与技术专业核心课程教学实施方案”展开研讨，并于2009年制定出包括“数据结构与算法”在内的8门核心课程的教学实施方案。方案按课程和培养类型分别列出详细的教学内容大纲，知识矩阵，学时分配，各知识点的讲授提示，重点和难点，实验内容和要求，考核内容、形式和要求，以及对教材写作的要求和建议。

作者参与制定了“数据结构与算法”（应用型）课程教学实施方案，并按照要求，精心为广大读者打造出这本内容丰富、适用面广、易学易用、具有特色的教材。“注重基础，强调实践，开发智力，培养能力，提高素质”是本书的写作宗旨。

首先，本书紧扣方案的主导思想，涵盖方案要求的全部知识点。考虑到不同学校教学要求上的差异和培养计算机专业应用型人才的实际需求，本着“有用、够用、实用”的原则，对某些知识点进行了适当的扩充和提高。这些内容一部分可用于选讲，一部分可作为学生课外阅读，以进一步改善学生的知识结构，扩大知识面。

其次，在突出重点、有效化解难点方面做了认真的考虑和合理的安排。对于那些既是重点又是难点的内容（比如，如何建立链表中链的概念以及遍历二叉树的递归概念），不惜加重笔墨；而对于那些非重点的难点，有的只介绍大致“轮廓”（如文件的组织结构），有的只做“科普式”的介绍（比如问题固有难度）。对于基本运算的算法，都给出C/C++实现程序（这些程序全部在Visual C++ 6.0环境下上机通过）；而对于那些实现程序过于冗长的非基本运算（如AVL树、红黑树，以及B树和B+树的处理），只给出算法的描述形式，而将算法的实现程序作为提高性的课后练习。将数据结构的类实现方法放在全书的最后也是出于这一考虑。因为用类去封装数据结构，如同给数据结构穿上一层厚厚的“铠甲”，不仅涉及的语法概念过于繁杂，初学者很难理解，而且需要透过“铠甲”才能看到算法的核心处理步骤，与其让学生带着“铠甲”训练，不如先轻装学透数据结构的处理算法，之后再学习如何“穿戴铠甲”。这样，既有效地化解了教学难点，也为学习面向对象的程序设计技术作了铺垫。

第三，以查找、插入、删除运算为主线介绍表、树、图等基本数据结构的特点、存储方法、时空效率。因为，查找、插入、删除不仅是最基本的、最常用的运算，而且往往也是不可分的

（通常联合使用，极少单独使用）。将这三种运算构成的运算集作为一个整体，可以得出结构的整体时空效率。正因为如此，本书未将“查找”作为一个独立问题单用一章集中介绍，而是将其作为基本结构的常用运算之一，与结构的其他运算一起讨论。

第四，作者力图通过本书为读者构建多层面全方位的知识体系。具体地说，在问题层面上，揭示问题的固有难度和 NP 难题的存在，使读者不至于为试图谋求优于固有难度的算法（尤其是为 NP-完全问题谋求有效算法）而做无用功（或说外行话）；在算法层面上，通过算法复杂性和评价方法，使读者了解什么是“有效算法”，什么是“好”算法，什么是“最优”算法，以及（通过学习算法设计的一般方法）如何设计出“好”算法；在数据结构层面，介绍各种数据结构的特点以及如何合理地为算法配置适当的数据结构，以获取预期的处理效率；在程序设计层面上，通过大量的示例算法的实现程序，向读者展示程序设计技术，从而将算法、数据结构、程序设计有机地融合在一起。

第五，精心设计的题量大、题型多、考查面宽、具有一定难度层次分布的 400 多道练习题，足以构建一个中等规模的习题库。题型包括基本概念题、普通填空题、程序填空题、选择题、算法设计题、基础实验题、综合实验题等。习题的考查面覆盖书中所有主要知识点。对重点内容配备的习题不但题量大、题型多，而且更偏重于编程练习（包括算法设计题、基本实验题和综合实验题），以帮助学生充分消化吸收所学内容。这 400 多道习题中，一般难度的约占 30%，中等难度的约占 40%，稍高难度的约占 20%，难度较大的约占 10%，教师可以针对不同的教学对象和不同的教学要求，选配不同难度的习题。通过多题型、多难度层次习题的综合训练（尤其是上机练习），可以有效地加深学生对基本概念的理解，强化学生的程序设计能力。

此外，附录B给出了一部分习题的参考答案（给出答案的题都是一些具有启发性的典型题），既便于教师备课与授课，又便于学生自学自测和复习。正文中带波浪线的词在附录 A 给出了中英文对照。

总之，在整体结构，内容取舍、主次分布、重点和难点，习题配备等方面都做了周全的考虑。既考虑到了由易逐步过渡到难，也照顾到了知识的系统性和完整性。

全书共分为 8 章。第 1 章概括介绍数据结构和算法的概念以及算法的描述方法和评价方法。第 2～4 章分别介绍基本数据结构——表树图。第 5 章介绍常用集合运算的算法设计和效率分析。第 6 章介绍基本排序方法（内排序和外排序）。第 7 章用通俗易懂的语言介绍问题的固有难度和 NP-完全问题的概念，以及常用的算法设计方法。第 8 章介绍基本数据结构的类实现方法。其中，第 1～4 章、第 6 章和第 5 章中的散列表是基本内容，而第 2～4 章和第 6 章是重点内容。标题带“*”的章节是可选讲内容（偏难的或非重点的）。

各校根据实际教学要求和学时分配，可以适当地对本书内容进行节选。建议按照下述方式进行教学内容的取舍。

第一层次（最高要求）：全面完成本书内容，但可以将第二层次忽略的内容列为课外阅读（不占学时）。

第二层次（较高要求）：可以忽略 2.5.3、2.7.4、5.2.3、5.4.5、6.7.3、6.7.4、6.7.5、7.2、7.3.4

各节内容。

第三层次（普通要求）：可以再忽略 1.2.3、2.5.2、2.7.3、2.9、4.2.4、4.4、4.7.3、5.3、5.4.3、5.4.4、5.5.2、6.5.1、6.7、7.1、7.3.3 各节和第 8 章的内容。

第四层次（最低要求）：可以在第三层次的基础上再忽略 2.5、2.6.2（其中的矩阵乘法内容）、2.8、3.2.2、3.2.5、3.5、4.6.2、5.4.1、7.3 各节的内容。

陈卫卫制定了全书的章节结构和内容编选原则，并编写了第 1、2、3、8 章；第 4～7 章由王庆瑞编写。

特别感谢蒋宗礼教授认真详尽地审阅了本书，并提出了宝贵的修改意见。

本书在编写过程中，一直得到高等教育出版社、解放军理工大学的关注和支持，在此一并致谢。

本书有配套的 PPT 课件，可以从中国高校计算机课程网（http://computer.cncourse.com）下载。

对于书中不当甚至错误之处，恳请读者批评、指正。

作　者

2010 年 9 月

目 录

第1章
概　述

1.1　基本概念

1.1.1　数据结构的概念

1．数据和数据结点

在计算机领域中，**数据**是对客观事物的描述形式和编码形式的统称，是计算机算法和程序的处理对象和计算结果。

在研究算法时，着重考虑由众多相同类型数据组成的数据集合，以及集合中数据之间的内在联系，需要对数据和数据集合进行哪些运算，以及如何提高运算效率，设计出高质量的程序。

数据元素也称为数据结点，简称**结点**。通常，一个数据结点由用来描述一个独立事物的名称、数量、特征、性质的一组相关信息组成。多数情况下，一个结点含有多个**数据项**，每个数据项是结点的一个**域**，能够用来唯一标识结点的域称为**关键字**域，人们往往用关键字指明结点。当然，某些情况下，一个结点只含一个数据项，即单值结点。

例如，在设计处理学生成绩问题的程序时，一名学生的相关信息，包括学生的姓名、学号、各科考试成绩等，构成一个数据结点，学号可以作为关键字。在处理库存商品问题时，一个数据结点对应一种商品的相关数据项，包括商品编号、名称、规格、数量、生产厂家、单价、入库日期等，商品编号可以作为关键字。在后面的示例中，多选用单值结点，也可理解成用关键字值代替结点值。

2．数据结构的定义

一个有穷的结点集合 D，以及该集合中各结点之间的关系 R，组成一个**数据结构**，表示成 $B=(D, R)$[①]。

这里，D 和 R 是对客观事物的抽象描述，R 表示结点之间的逻辑关系，所以，(D, R)指的是数据的**逻辑结构**。

---

① 数据结构的另一种定义形：$B=(D, R, O)$，其中“O”表示“运算集”。

例如，D={a，b，c，d，e}，R_1={<a，b>，<d，c>，<b，d>，<c，e>}，R_2={<a，b>，<b，c>，<b，d>，<b，e>，<d，e>}。那么，(D，R_1)组成一个数据结构；而(D，R_2)则组成另一个数据结构。

数据结构在计算机内的存储形式称为存储结构，也称为**物理结构**。在存储数据结点值的同时，还必须存储结点之间的关系（或结点的存储地址能够体现出结点之间的关系）。用来存储一个数据结点，并在必要时存储该结点与其他结点之间关系的一组存储单元称为一个存储结点。因为一个数据结点对应一个存储结点，所以，在不会混淆的情况下，也把存储结点简称为结点。当前尚未存储数据结点的存储结点叫空白结点，或称空结点、自由结点。

例如，n 个数据结点的集合 D={a_0，a_1，a_2，…，a_{n-1}}，关系 R={<a_i，a_{i+1}>|i=0，1，…，n–2}。可以将 D 存储于数组 $b[m]$（n<m）中，那么每个 $b[i]$（i=0，1，…，m–1）是一个存储结点，除 n 个数据元素占用的 n 个单元外，尚有 m–n 个空白结点。如果将 n 个数据元素依次存储于 $b[0]$～$b[n-1]$，那么结点的存储地址（即数组元素下标）能够体现出结点之间的关系，关系 R 用不着单独存储。如果将 n 个数据元素随机地存储于 n 个单元，除存储数据元素 a_i 外，还要存储能够体现关系<a_i，a_{i+1}>的信息。通常的做法是为存储结点增加一个域，若 a_i 存储于 $b[j]$，那么对于 $b[j]$来说，增加的这个域则用于存储元素 a_{i+1} 的地址。

3. 数据结构的种类

基本数据结构大体上可分成表结构、树结构、图结构和散结构 4 大类，如图 1-1 所示。

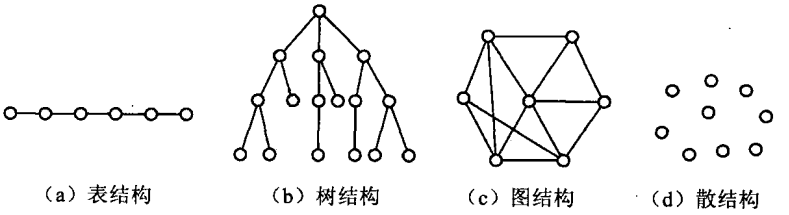


图 1-1 数据结构示意图

表结构用来表示结点之间某种先后次序关系（属于线性关系），一个结点只有一个前驱和一个后继，但首结点没有前驱，尾结点没有后继。比如学生成绩单。

树结构用来表示结点之间的层次关系、分支关系和嵌套关系，一个结点只有一个前驱（根结点没有前驱），但可以有多个后继。比如某部门的组织机构、行政区划图、文件目录结构。

图结构用来表示结点之间最一般的关系，任何两个结点之间都可能有（或没有）关系。比如城市交通网。

通常认为，表结构描述的是“一对一”关系，树结构描述的是“一对多”关系，而图结构描述的是“多对多”关系。

如果集合中任何两个结点之间都没有关系，或者说存在一种特殊的关系——无关关系，则可表示成散结构。

从数学角度来讲，树结构和散结构都是图的特例，表结构又是树的特例。例如，无回路的

连通图就是树，散结构对应的是只含顶点而没有边的“空图”。线性表也可以看作“一元树”。

1.1.2 抽象数据类型

1．抽象的意义

从一般意义上讲，抽象是指“抽取事物的共性，忽略个性；体现外部特征，掩盖具体细节”。

在计算机程序设计领域中，“抽象”起到十分关键的作用。在系统分析和设计阶段，通过对客观事物的抽象，能够让人们更好地把握全局，从更高一层的宏观角度考虑问题，着眼于制作更具有生命力的软件模块，以增强软件的可复用性。同时，在编程实现阶段，通过抽象数据类型实现“信息隐蔽”，从而使软件具有易移植和易维护的特点。

抽象程度越高，对信息及其处理细节的隐藏就越深，对使用这些信息的人们的计算机专业知识要求也就越低，使用起来就越方便。

最初，汇编语言的出现，使人们能够像数学那样直接使用形如 23、1.54、3.2E05 等数据，而不必关心数据在计算机中是如何表示的，如何进行计算的，其中的处理细节可能涉及定点数、浮点数的二进制存储形式以及运算方式，这些细节都由汇编程序自动完成，对使用者来说是隐蔽的，或者说是透明的。

其后，高级程序设计语言提供了整型、单/双精度实型、字符型等基本数据类型，以及这些类型的相应运算、函数库、输入输出语句。用户在使用这些类型的数据进行计算时，可以直接将它们写成表达式，系统便按照运算符的优先级，自动地对表达式进行加工处理，并给出计算结果。这中间究竟隐藏了多少细节，对于那些不懂得“编译”过程的人们是很难理解，也很难说清楚的，但是这种隐藏给程序设计者带来的方便却是实实在在的。

伴随着抽象不断地向高层扩展，还引发了程序设计方法的变革。从面向过程的程序设计，发展到面向对象的程序设计，再逐步发展到面向组件、面向服务的程序设计。这对软件复用技术的发展起到了十分关键的作用。

2．抽象数据类型的含义

抽象数据类型简称 **ADT**，是将“数据”连同对其的“处理操作”（即运算）封装在一起而形成的复合体。ADT 是对一个确定的数学模型，以及定义在该模型上的一组操作的抽象表示，不涉及具体的实现。

例如，可以将与有序表有关的数据和处理操作封装成一个 ADT，涉及的数据可能有元素个数、数据元素等，涉及的操作可能有查找、插入、删除等，其描述如下：

```
ADT    Orderlist          //Orderlist 为抽象数据类型的名字
{
  数据对象：D={a_i|a_i∈ElemType，i=1,2,…,n，n≥0}
  数据关系：R={<a_i-1,a_i>| a_i-1,a_i∈D, a_i-1≤a_i，i=2,3,…,n}
```

```
基本操作：
 InitList(&L)              //构造一个空的有序表 L
 InsertElem(&L，e)         //在有序表中插入新元素 e，使表仍有序
 DeleteElem(&L，e)         //在有序表中删除元素 e
 ListTraverse(L)           //输出表中的元素
 ListSearch(L，e)          //在表中查找值为 e 的元素
 ListLength(L)             //返回表中元素的个数
 ListEmpty(L)              //测试表空否
}
```

3．抽象数据类型的实现方法

实现抽象数据类型就是要编写相应的程序，实现 ADT 所定义的功能。具体地说，就是确定数据的类型和存储结构，使其能够准确地体现 ADT 中定义的数据对象和数据关系，并在此基础上，为 ADT 中定义的每一个运算设计求解算法，并编写实现函数。

实现 ADT 可采用封装法、半封装法和分散法。

封装法将 ADT 中定义的数据和处理数据的函数封装成一个整体，比如 C++的类，于是一个 ADT 对应一个类。还可以通过继承，派生出具有个性的子类。

半封装法将 ADT 数据和为处理数据需要而定义的相关变量封装在一起形成一个结构，而将有关的处理函数定义在结构体之外。

分散法则将数据和处理数据的函数各自分开定义。

3 种方法相比，封装法与 ADT 的定义格式较为一致，数据和处理函数的归属明确，也比较符合“面向对象的程序设计方法”的要求。

分散法有可能将属于 ADT 的和不属于 ADT 的数据、函数混杂在一起，尤其是同一个程序涉及多个 ADT 时，不同 ADT 所属的数据和函数的混杂，使得无法从程序的物理结构上（即代码的物理次序）区分哪些数据和函数属于哪个 ADT。

半封装法仅做到对数据存储结构的封装（请参见 2.1.2 节），其特点介于封装法和分散法之间。

由于本书前 7 章着重介绍基本数据结构的特点、存储方式和处理算法，而没有刻意地通过 ADT 描述数据结构，又由于同一个问题所涉及的数据和处理算法相对独立、归属明确，即便定义 ADT，也不会同时涉及多个 ADT，所以没有采用封装法，而是采用分开描述、分开定义的方法。更为主要的是，这样做能够简化程序结构，减少篇幅，突出算法的核心步骤，便于阅读、学习和掌握。

最后用一章的篇幅（第 8 章）集中介绍如何将数据和处理数据的函数进行封装，也就是用 C++的类实现前几章中介绍的各种数据结构及其处理算法，从而体现出数据结构的面向对象的

实现方法。

1.1.3 算法的概念

1. 算法的定义

计算机科学大师克努特（Donald E. Knuth）在他的《计算机程序设计艺术》（第 1 卷：基本算法）中，给出了算法的定义，其要点如下：

算法是有穷规则的集合，而规则规定了解决某一特定问题的运算序列，同时，算法还应具有“有穷性、确定性、可行性以及输入数据、输出数据”。

- 有穷性——必须在执行有穷步骤后终止。
- 确定性——每一步必须具有确切的定义，不能含混不清，无二义性。
- 可行性——所有的运算都可以精确地实现。
- 必须有输出数据，包括输出某种动作或控制信号。没有任何输出的算法相当于“什么也没做”。
- 往往要求输入数据，作为执行前的初始量。但有的算法不要求输入数据。

2. 算法、数据结构与程序的关系

当人们着手求解某给定问题，设计求解此问题的处理程序时，先要确定方案，设计求解算法，然后为算法选择满足处理要求的数据结构，将算法的计算步骤分解成对数据结构的运算，再设计实现数据结构运算的算法，最终编写求解程序。程序就是问题的最终解，称为问题的算法解，或程序解、软件解。

其间，还要对算法的性能进行分析，主要是论证算法的正确性、评估算法的运行效率。若分析结果不能满足要求，则要重新设计算法，或对算法进行改进，并再次分析。直到分析结果能够满足解题要求，再进行后续的编程工作。在交付使用之前，还要适当地选择一些数据，对程序进行调试和测试。

按算法编写程序的过程称为对算法的实现。程序不仅要“完全忠实于”算法，不折不扣地实现算法的处理步骤，而且要确保算法能够达到预期的运行效率。其中，数据结构的选取是至关重要的，它不仅影响算法实现的难易，而且会影响算法的运行效率。著名的 Wirth 公式“算法+数据结构=程序”就指出了这一道理。

例如，Kruskal 算法（请参见 4.5.1 节）的实现就颇费了一番脑筋。再如，对于单一的集合合并运算来说，链式存储要比顺序存储好得多。

于是，算法有两种表现形式：描述形式和程序形式。描述形式是算法的原始形式；程序形式是算法的实现形式，也是算法的“最终描述形式”。因为程序中含有算法，而算法又是程序的原型。所以，后文有时将程序称为算法，或将算法称为程序。

1.2 算法的描述和评价

1.2.1 算法的描述

1. 描述形式

算法的描述形式依赖于描述语言。描述语言可以是自然语言，即汉语或英语等人们日常生活中使用的语言，还可以是专为描述算法而设计的“格式化”语言，即人工语言。

描述算法时，往往为了突出算法的主体，而忽略支持算法执行的某些有碍于阅读的具体细节，以便于理解、记忆和交流。

目前，常用的描述语言主要有自然语言、流程图、类程序设计语言。下面简单介绍这3种描述语言的用法和特点，并给出示例。

自然语言描述形式也称为文字描述形式或文字叙述形式，是算法最原始的表现形式。这种形式直接记录了人们求解问题的思维过程，简单直观。当然，为便于理解和阅读，最好采用分步骤的叙述方式，将处理步骤描述得更具条理性和准确性。

用流程图描述算法时，要注意使用规范的流图符号（见图1-2），例如，矩形框用于书写处理步骤，菱形框用于书写判断条件，椭圆形框用于表示开始和终止。

用流程图描述大型程序时，通常按描述的精细程度分成若干级，先画粗框，再逐步细化。

类程序设计语言（简称类语言）是以某种通用的程序设计语言作为“宿主语言”，对其略加改造而形成的一种“非正规的”语言。比如类C语言、类C++语言、类Java语言、类Pascal语言等都是常用的算法描述工具。用类语言书写的程序称伪程序或伪代码。

为了描述方便，伪程序中的语句可以不那么严格，只要混杂的语句大家都能理解，以屏蔽某些处理细节（比如进栈退栈操作细节等）。尤其不要求带有大段大段的类型定义、变量定义、输入输出等对算法只起“簿记性”工作的语句。只保留程序的基本框架结构，突出算法中的计算性语句，以及循环、递归、函数调用等主要控制性语句。

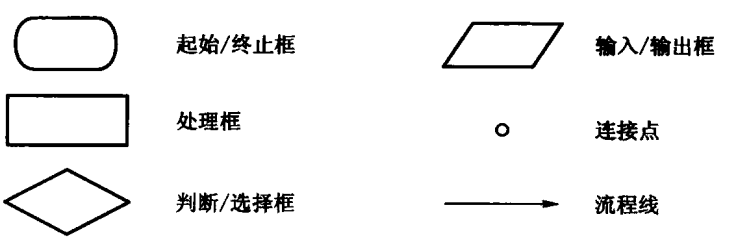


图1-2 常用的流程图符号