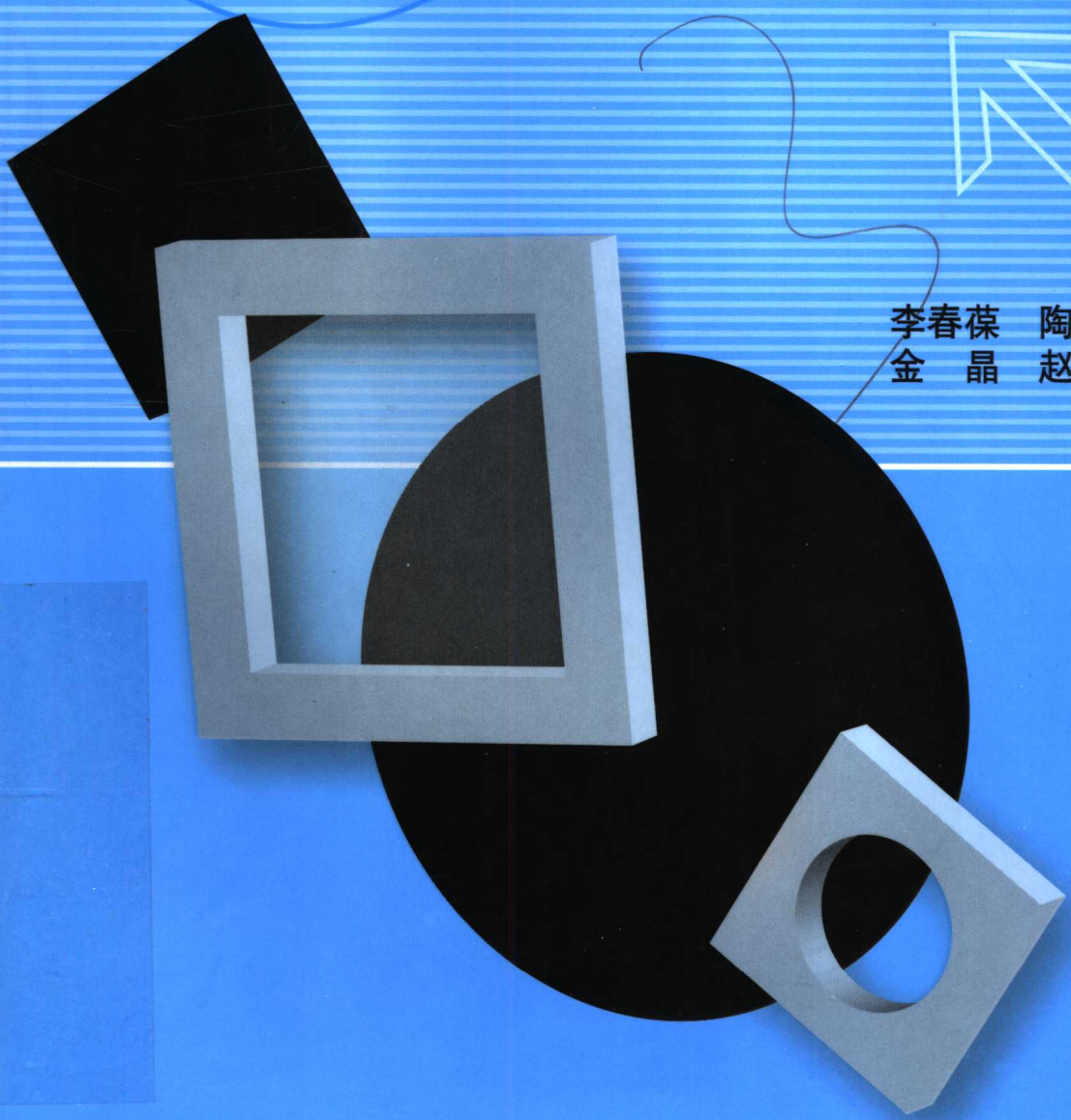
计算机与信息技术专业应用教材

数据结构与算法教程

（第2版）



李春葆　陶红艳
金　晶　赵丙秀　编著

清华大学出版社

▶ 计算机与信息技术专业应用教材

数据结构与算法教程（第2版）

李春葆　陶红艳　金晶　赵丙秀　编著

清华大学出版社
北京

内 容 简 介

数据结构是计算机专业的核心课程，主要传授数据组织方法和典型问题求解策略，具有一定的抽象性，不易掌握。本书是《数据结构与算法教程》的第2版，内容安排更加合理，讲解更加流畅。

本书作者具有多年授课经验，对教学重点和学习难点有深刻了解。在内容安排上，以教学大纲为指导，充分考虑课程特点，兼顾学习习惯。全书分为11章，内容涉及数据结构的基本概念、线性表、栈和队列、串和数组、递归和广义表、树和二叉树、图、查找、内排序、外排序、文件以及算法设计技术。

书中精心设计大量例题，用于演示说明相关概念和方法；各章在课后都给出多个典型练习题，并在附录中提供参考答案。其目的是加深理解，强化应用。

本书适合用作高等院校相关专业“数据结构”课程的教学用书。

图书在版编目（CIP）数据

数据结构与算法教程/李春葆，陶红艳，金晶，赵丙秀编著. —2 版

—北京：清华大学出版社，2007.10

ISBN 978-7-302-16110-3

Ⅰ. 数… Ⅱ. ①李… ②陶… ③金… ④赵… Ⅲ. ①数据结构—高等学校—教材 ②算法分析—高等学校—教材 Ⅳ. TP311.12

中国版本图书馆 CIP 数据核字（2007）第 141464 号

责任编辑：刘秀青
责任校对：科　海
责任印制：科　海
出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
c - service@tup.tsinghua.edu.cn
社 总 机：010-62770175　　**邮购热线**：010-62786544
投稿咨询：010-62772015　　**客户服务**：010-82896445
印 装 者：北京市鑫山源印刷有限公司
经　　销：全国新华书店
开　　本：787×1092　1/16　　**印张**：19.75　　**字数**：481 千字
版　　次：2007 年 10 月第 2 版　　2007 年 10 月第 1 次印刷
印　　数：0001~4000
定　　价：29.00 元

前　言

当用计算机来解决实际问题时，就要涉及到数据的组织及数据的处理，而数据组织及数据处理正是数据结构课程的主要研究对象。

数据结构与算法课程在计算机科学中是一门综合性的专业基础课。在计算机科学中，数据结构与算法不仅是一般程序设计的基础，而且还是设计和编译程序、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。

数据结构与算法主要研究内容有：数据的逻辑结构，即数据关系之间的逻辑关系；数据的存储结构，即数据的逻辑结构在计算机中的表示；操作算法，即插入、删除、修改、查询、排序等。

作者针对该课程的特点，总结长期教学经验，于2005年6月出版了《数据结构与算法教程》，获得了教师与学生的好评。本书在原版书的基础上，补充了新内容和新实例，修改了部分章节，使内容符合时代的发展，章节安排更加合理，讲解更加顺畅。

全书分为11章。第1章为概论，介绍数据结构的基本概念，特别强调算法分析的方法；第2章为线性表，介绍线性表的两种存储结构即顺序表和链表的逻辑结构与基本运算的实现过程；第3章为栈和队列，介绍这两种特殊的线性结构的概念与应用；第4章为串和数组，介绍串的概念、模式匹配算法、多维数组和稀疏矩阵的基本运算；第5章为递归和广义表，介绍广义表的概念与递归运算算法；第6章为树和二叉树，介绍树和二叉树的概念与各种运算，特别突出递归算法的实现过程；第7章为图，介绍图的概念树与图的各种运算的实现过程；第8章为查找，介绍各种查找算法的实现过程；第9章为内排序，介绍各种内排序算法的实现过程；第10章为文件，介绍各种文件组织方式和外排序算法；第11章为算法设计技术，介绍常用的算法及其在数据结构中的应用。

本书适合于作为计算机及相关专业“数据结构”课程的教材，也适合于计算机水平考试人员参考。

由于水平所限，尽管编者不遗余力，仍可能存在错误和不足之处，敬请读者批评指正。

编　者

2007年10月

目　录

第1章　概论 1

1.1　什么是数据结构 1
1.1.1　逻辑结构 2
1.1.2　存储结构 4
1.1.3　数据运算 7
1.1.4　数据结构和数据类型 7
1.1.5　程序=数据结构+算法 7
1.2　算法和算法分析 8
1.2.1　算法及其表示 8
1.2.2　算法分析 9
练习题1 11

第2章　线性表 13

2.1　线性表的基本概念 13
2.1.1　线性表的定义 13
2.1.2　线性表及其基本运算 14
2.2　线性表的顺序存储结构 15
2.2.1　顺序表 15
2.2.2　线性表基本运算在顺序表上的实现 16
2.2.3　顺序实现的算法分析 18
2.2.4　顺序表的应用示例 19
2.3　单链表存储结构 21
2.3.1　单链表 21
2.3.2　线性表基本运算在单链表上的实现 21
2.3.3　循环单链表 28
2.4　双链表存储结构 32
2.4.1　双链表 32
2.4.2　线性表基本运算在双链表上的实现 32
2.4.3　循环双链表 35
2.5　链表的应用 39
练习题2 44

第3章　栈和队列 46

3.1　栈 46
3.1.1　栈的基本概念 46
3.1.2　栈的顺序存储结构 48
3.1.3　栈的链式存储结构 51
3.1.4　栈的应用示例 54
3.2　队列 56
3.2.1　队列的基本概念 56
3.2.2　队列的顺序存储结构 57
3.2.3　队列的链式存储结构 61
3.2.4　队列的应用示例 64
练习题3 66

第4章　串和数组 67

4.1　串 67
4.1.1　串的定义 67
4.1.2　串的顺序存储结构及其基本运算实现 68
4.1.3　串的链式存储结构及其基本运算实现 72
4.1.4　串的模式匹配 77
4.2　数组 81
4.2.1　数组的定义 81
4.2.2　数组存储的排列顺序 82
4.2.3　数组基本运算的实现 82
4.2.4　特殊矩阵的压缩存储 83
4.3　稀疏矩阵 85

4.3.1 稀疏矩阵的三元组表示 85
4.3.2 稀疏矩阵的十字链表表示 90
练习题4 91

第5章 递归和广义表 93

5.1 递归 93
5.1.1 什么是递归 93
5.1.2 如何设计递归算法 94
5.2 广义表的定义 99
5.3 广义表的存储表示 99
5.4 广义表的基本运算算法 101
5.5 广义表的递归算法 107
练习题5 109

第6章 树和二叉树 111

6.1 树 111
6.1.1 树的定义 111
6.1.2 树的表示 112
6.1.3 树的基本术语 113
6.1.4 树的存储结构 114
6.2 二叉树 115
6.2.1 二叉树的定义 115
6.2.2 二叉树的性质 116
6.2.3 二叉树的存储结构 118
6.3 二叉树的基本运算算法 120
6.3.1 二叉树的基本运算 120
6.3.2 二叉树基本运算实现算法 120
6.4 二叉树的遍历 125
6.4.1 常用的二叉树遍历算法 125
6.4.2 遍历算法的应用 128
6.5 二叉树与树之间的转换 130
6.5.1 树转换成二叉树 130
6.5.2 森林转换为二叉树 131
6.5.3 二叉树还原为树或森林 132
6.6 线索二叉树 133
6.6.1 线索 133
6.6.2 线索二叉树的存储结构 133
6.6.3 二叉树的线索化 135
6.6.4 线索二叉树的基本运算算法 136
6.7 哈夫曼树 138
6.7.1 哈夫曼树的定义 138
6.7.2 构造哈夫曼树 139
6.7.3 哈夫曼编码 141
练习题6 143

第7章 图 145

7.1 图的基本概念 145
7.1.1 图的定义 145
7.1.2 图的基本术语 146
7.2 图的存储结构 148
7.2.1 邻接矩阵 149
7.2.2 邻接表 151
7.3 图的遍历 154
7.3.1 广度优先搜索 154
7.3.2 深度优先搜索 155
7.3.3 图遍历算法的应用 156
7.4 最小生成树 160
7.4.1 普里姆算法 160
7.4.2 克鲁斯卡尔算法 164
7.5 最短路径 166
7.5.1 单源最短路径 166
7.5.2 每对顶点之间的最短路径 169
7.6 拓扑排序 173
7.7 AOE网与关键路径 175
练习题7 177

第8章 查找 180

8.1 顺序查找 180
8.2 二分查找 182
8.3 分块查找 184

8.4 二叉排序树 186
8.4.1 二叉排序树的定义 186
8.4.2 二叉排序树的基本运算 187
8.5 二叉平衡树 191
8.6 哈希表查找 195
8.6.1 哈希表查找的基本概念 195
8.6.2 构造哈希函数的方法 196
8.6.3 哈希冲突解决方法 197
练习题8 202

第9章 内排序 203

9.1 排序的基本概念 203
9.2 插入排序 203
9.2.1 直接插入排序 204
9.2.2 希尔排序 205
9.3 选择排序 207
9.3.1 直接选择排序 207
9.3.2 堆排序 208
9.4 交换排序 211
9.4.1 冒泡排序 211
9.4.2 快速排序 212
9.5 归并排序 214
9.6 基数排序 217
练习题9 219

第10章 文件 221

10.1 概述 221
10.2 文件组织 221
10.2.1 顺序文件 221
10.2.2 索引文件 222
10.2.3 哈希文件 223
10.2.4 多关键字文件 224
10.3 动态索引 226
10.3.1 B-树的定义 226
10.3.2 B-树的查找 226
10.3.3 B-树的插入 227
10.3.4 B-树的删除 228
10.3.5 B+树 230
10.4 外排序 233
10.4.1 排序过程 233
10.4.2 多路平衡归并 234
10.4.3 初始归并段的生成 236
10.4.4 最佳归并树 238
练习题10 240

第11章 算法设计技术 241

11.1 迭代法 241
11.2 穷举法 244
11.3 递归法 246
11.4 回溯法 250
11.5 分枝限界法 260
11.6 分治法 262
11.7 动态规划法 263
练习题11 266

附录A 习题参考答案 267

附录B 本书算法中使用的C/C++语法说明 304

参考文献 306

第1章 概 论

CHAPTER 01

计算机的功能主要是处理数据，这些数据绝不是杂乱无章的，而是有着某种内在联系。只有分清楚数据的内在联系，合理地组织数据，才能对它们进行有效的处理。如何合理地组织数据、高效率地处理数据，正是本书的目的。本章简要介绍有关数据结构的基本概念和算法分析方法。

1.1 什么是数据结构

数据是信息的载体，能够被计算机识别、存储和加工处理。数据包括文字、表格、图像等。例如，一个班的全部学生记录、a～z的字母集合、1～1000之间的所有素数等都是数据。

数据元素（有时称为结点、记录等）是数据的基本单位，在程序中通常把它作为一个整体进行处理。例如，一个班的学生包括张三、李四等数据元素。

有时一个数据元素可以由若干个数据项(也可称为字段、域、属性)组成。数据项是具有独立意义的不可分割的最小标识单位。如整数这个集合中，10这个整数就是一个数据元素。又比如在一个数据库（关系数据库）中，一个记录可称为一个数据元素，而这个元素中的某一字段就是一个数据项。

数据对象是具有相同类型的数据元素的集合。

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。这些数据元素不是孤立存在的，而是有着某种关系，这种关系称为结构。数据结构一般包括以下3个方面的内容：

（1）数据元素之间的逻辑关系，所有元素的逻辑关系构成了逻辑结构。数据的逻辑结构是从逻辑关系上描述数据，它与数据的存储无关，是独立于计算机的。因此，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。

（2）数据元素及其逻辑关系在计算机存储器内的表示，构成了数据的存储结构；数据的存储结构是逻辑结构用计算机语言的实现(亦称为映像)，它是依赖于计算机语言的。一般只在高级语言的层次上来讨论存储结构。

（3）数据的运算，即对数据施加的操作。数据运算的定义（指定运算的功能）是基于

逻辑结构的，每种逻辑结构都有一组相应的运算。例如，最常用的运算有：检索、插入、删除、更新、排序等。数据运算的实现是基于存储结构的，是采用某种计算机语言编写的算法。

通常情况下，同一种逻辑结构可以设计多种存储结构，在不同的存储结构中，实现同一种运算的算法可能不同。

逻辑结构、存储结构和运算3者之间的关系如图1.1所示。

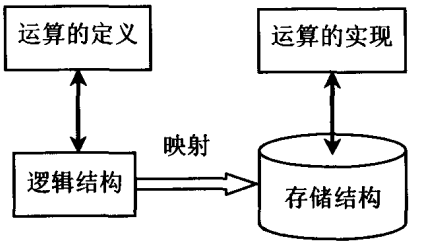


图 1.1　逻辑结构、存储结构和运算之间的关系

1.1.1 逻辑结构

数据元素之间的逻辑关系称为数据的逻辑结构。根据数据元素之间逻辑关系的不同特性，分为下列4类基本结构。

- ◈ 集合：结构中的数据元素同属于一个集合（集合类型由于元素之间的关系过于松散，数据结构课程中较少讨论）。
- ◈ 线性结构：结构中的数据元素存在一对一的关系。
- ◈ 树形结构：结构中的数据元素存在一对多的关系。
- ◈ 图形结构：结构中的数据元素存在多对多的关系。也称为网状结构。

在大多数情况下，数据的逻辑结构可以用二元组

```
S=(D,R)
```

来表示。其中D是结点（即数据元素）的有限集合，即D是由有限个结点所构成的集合；R是D上的关系的有限集合，即R是由有限个关系所构成的集合，而每个关系都是从D到D的关系。

在表示每个关系时，用尖括号表示有向关系，如<a,b>表示存在结点a到结点b之间的关系；用圆括号表示无向关系，如（a,b）表示既存在结点a到结点b之间的关系，又存在结点b到结点a之间的关系。设r是一个D到D的关系，r∈R，若d，d'∈K，且<d,d'>∈r，则称d'是d的后继结点，d是d'的前趋结点，这时d和d'是相邻的结点（都是相对r而言的）；如果不存在一个d'使<d,d'>∈r，则称d为r的终端结点；如果不存在一个d'使<d',d>∈r，则称d为r的开始结点；如果d既不是终端结点也不是开始结点，则称d是内部结点。

例如一个城市表，如表1.1所示，就是一个数据结构，它由很多记录（这里的数据元素就是记录）组成，每个元素又包括多个字段（数据项）。那么这个表的逻辑结构是怎么样的呢？通常数据结构都是从数据元素之间的关系来分析，对于这个表中的任一个记录，它

只有一个前趋结点，也只有一个后继结点，而且整个表只有一个开始结点和一个终端结点。由此可知，这个表的逻辑结构为线性结构。其逻辑结构表示如下：

```
City=(D,R)
D={Beijing,Shanghai,Wuhan,Xian,Nanjing}
R={r}
r={<Beijing,Shanghai>,<Shanghai,Wuhan>,<Wuhan,Xian>,<Xian,Nanjing>}
```

表 1.1　城市表

城市	区号	说明
Beijing	010	首都
Shanghai	021	直辖市
Wuhan	027	湖北省省会
Xian	029	陕西省省会
Nanjing	025	江苏省省会

数据的逻辑结构可以用相应的关系图来表示，称之为逻辑结构图。

【例1.1】　设数据的逻辑结构如下：

```
B1=(D,R)
D={1,2,3,4,5,6,7,8,9}
R={r}
r={<1,2>,<1,3>,<3,4>,<3,5>,<4,6> ,<4,7>,<5,8>,<7,9>}
```

试画出对应的逻辑结构图，并指出哪些是开始结点，哪些是终端结点，说明是何种数据结构。

解：B1对应的逻辑结构图如图1.2所示。其中，1是开始结点；2，6，8，9是终端结点。它是一种树形结构。

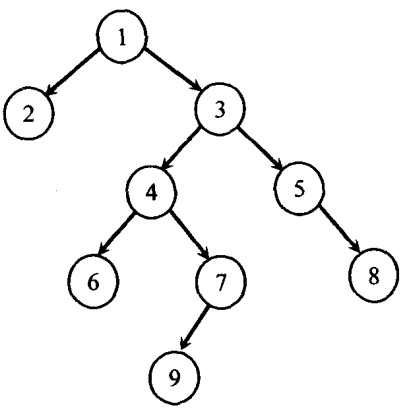


图 1.2　逻辑结构图

1.1.2 存储结构

数据对象在计算机中的存储表示称为数据的存储结构，也称为物理结构。

把数据对象存储到计算机中时，通常要求既要存储各数据元素的数据，又要存储数据元素之间的逻辑关系。在实际应用中，数据的存储方法是灵活多样的，可根据问题规模（通常是指元素数目的多少）和运算种类等因素适当选择。本书将介绍4种基本的存储结构：顺序存储结构、链式存储结构、索引存储结构和散列存储结构。下面简要介绍这些存储结构的特点。

1．顺序存储结构

顺序存储结构是把逻辑上相邻的元素存储在一组连续的存储单元中，其元素之间的逻辑关系由存储单元地址间的关系隐含表示，也就是说，顺序存储结构将数据的逻辑结构直接映射到存储结构。

对于前面的逻辑结构City，假定每个元素占用30个存储单元，数据从100号单元开始由低地址向高地址方向存储，对应的顺序存储结构如图1.3所示。

地址	城市名	区号	说明
100	Beijing	010	首都
130	Shanghai	021	直辖市
160	Wuhan	027	湖北省省会
190	Xian	029	陕西省省会
210	Nanjing	025	江苏省省会

图 1.3　顺序存储结构

顺序存储结构的主要优点是节省存储空间。因为分配给数据的存储单元全用于存放结点的数据值，数据元素之间的逻辑关系没有占用额外的存储空间。用这种方法来存储线性结构的数据元素时，可实现对各数据元素的随机访问。这是因为线性结构中每个数据元素都对应一个序号（开始元素的序号为1，它的后继元素的序号为2……），可以根据元素的序号i，计算出它的存储地址：

$$Loc(i)=q+(i-1)\times p$$

其中，p是每个元素所占的单元数；q是第一个元素所占单元的首地址。

顺序存储结构的主要缺点是不便于修改，在对元素进行插入、删除运算时，可能要移动一系列的元素。

2．链式存储结构

顺序存储结构要求所有的元素相邻存放，需占用一片连续的存储空间；而链式存储结构不是这样，它的每个结点单独存储，无需占用一整块存储空间。但为了表示结点之间的关系，需要给每个结点附加指针字段，用于存放相邻结点的存储地址。

假定给前面的逻辑结构City中的每个结点附加一个“下一个结点地址”，即后继指针

字段，用于存放后继结点的首地址，则可得到如图1.4所示的City的链式存储表示。从图中可以看出，每个结点占用两个连续的存储单元，一个存放结点的数值，另一个存放后继结点的首地址。

地址	城市名	区号	说明	下一个结点地址
100	Beijing	010	首都	210
130	Nanjing	025	江苏省省会	∧
160	Xian	029	陕西省省会	130
190	Wuhan	027	湖北省省会	160
210	Shanghai	021	直辖市	190

图 1.4 City 的链式存储表示

为了更清楚地反映链式存储结构，可采用更直观的图示来表示，如City的链式存储结构可用如图1.5所示的方式表示。

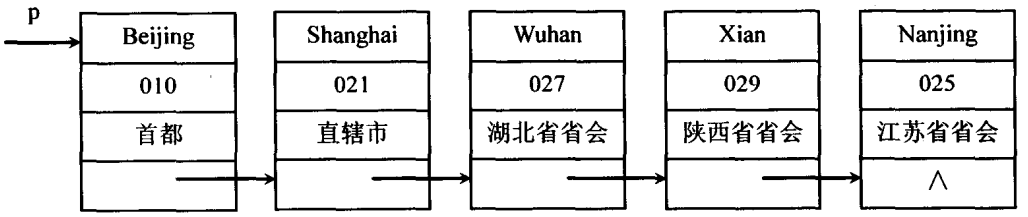


图 1.5 City 链式存储结构示意图

链式存储结构的主要优点是便于修改，在进行插入、删除运算时，仅需修改结点的指针字段值，不必移动结点。

与顺序存储结构相比，链式存储结构的主要缺点是存储空间的利用率较低，因为分配给数据的存储单元有一部分被用来存放结点之间的逻辑关系了。另外，由于逻辑上相邻的结点在存储器中不一定相邻，因此，在用这种方法存储的线性结构中不能对结点进行随机访问。

3. 索引存储结构

索引存储结构是在存储结点信息的同时，还建立附加的索引表。

索引表中的每一项称为索引项，索引项的一般形式是：

（关键字，地址）

关键字惟一标识一个结点，地址作为指向结点的指针。对于线性结构来说，各结点的地址在索引表中是按结点的序号依次排列的。图1.6是City的一种索引存储表示。

在进行关键字查找时，可以先在索引表中快速查找（因为索引表中按关键字有序排列，可以采用二分查找）到相应的关键字，然后通过地址找到结点表中对应的结点。

线性结构采用索引存储后，可以对结点进行随机访问。在进行插入、删除运算时，由于只需移动存储在索引表中的结点的存储地址，而不必移动存储在结点表中的结点的数值，

所以仍可保持较高的运算效率（这是因为，在一般情况下，结点中共包含有多个字段，移动一个结点的数值要比移动一个结点的地址花费更多的时间）。

索引存储结构的缺点是，为建立索引表而增加了时间和空间的开销。

索引表

地址	关键字	指针
300	010	100
310	021	130
320	025	210
330	027	160
340	029	190

结点表

地址	城市名	区号	说明
100	Beijing	010	首都
130	Shanghai	021	直辖市
160	Wuhan	027	湖北省省会
190	Xian	029	陕西省省会
210	Nanjing	025	江苏省省会

图 1.6 City 的索引存储表示

4. 散列存储结构

散列存储结构是根据结点的值确定结点的存储地址。具体做法是：以结点中某个字段的值为自变量，通过某个函数（称为散列函数）计算出对应的函数值i，再把i当作结点的存储地址。

对于City结构，假设以城市名的值作为自变量key，选用函数：

```
H(key)=ASC(LEFT(key,1)) mod 8
```

来计算结点的存储地址，其中，ASC(LEFT(key,1))表示取字符串key中第一个字符的ASCII码，mod是取模运算，计算结果如下：

key	Beijing	Shanghai	Wuhan	Xian	Nanjing
ASC(key)	66	83	87	88	78
H(key)	2	3	7	0	6

于是，可以得到如图1.7所示的City的散列存储表示。

地址	城市名	区号	说明
0	Xian	029	陕西省省会
1			
2	Beijing	010	首都
3	Shanghai	021	直辖市
4			
5			
6	Nanjing	025	江苏省省会
7	Wuhan	027	湖北省省会

图 1.7 City 的散列存储表示

散列存储的优点是查找速度快，只要给出待查找结点的数值，就有可能立即算出结点

的存储地址。

与前3种存储方法不同的是，散列存储方法只存储结点的数值，不存储结点与结点之间的逻辑关系。散列存储方法一般只用于要求对数据能够进行快速查找、插入的场合。采用散列存储的关键是要选择一个好的散列函数和处理“冲突”的办法。本书第8章将详细介绍这种存储方法。

在用高级语言编程时，可以用编程语言所提供的数据类型来描述数据的存储结构。例如，用“一维数组”表示一组连续的存储单元，来实现顺序存储结构、索引存储结构和散列存储结构；用C/C++语言中的“指针”来实现链式存储结构。

1.1.3 数据运算

数据运算就是施加于数据的操作。数据运算包括运算的定义和运算的实现，前者确定运算的功能，后者是在存储结构上确定对应运算算法。

在数据结构中，运算不仅仅是加减乘除这些算术运算，还常常涉及算法问题。算法的实现与数据的存储结构密切相关。

1.1.4 数据结构和数据类型

数据结构是指将按某种逻辑关系组织起来的一组数据元素，按一定的存储方式存储于计算机中，并在其上定义了一个运算的集合。

数据类型是高级程序设计语言中的一个基本概念，它和数据结构的概念密切相关。

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。类型明显或隐含地规定了数据的取值范围、存储方式以及允许进行的运算。因此可以认为，数据类型是在程序设计语言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，必须借助编程语言所提供的数据类型来描述数据的存储结构。

1.1.5 程序=数据结构+算法

著名的瑞士计算机科学家沃思指出：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，而算法是对数据运算的描述。因此，程序设计的实质是为实际问题选择一种好的数据结构，加之设计一个好的算法，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。所以说，选择合适的数据结构是解决应用问题的关键步骤。

一般地，设计一个应用程序时，要先分析问题，确定其中的主要数据的逻辑结构，并定义对应的基本运算。然后结合逻辑结构和基本运算设计出合适的存储结构，并在此基础上设计出实现基本运算功能的函数，最后应用程序调用这些基本运算函数完成特定的功能。如图1.8所示，这种程序设计方式体现了结构化程序设计的特点。

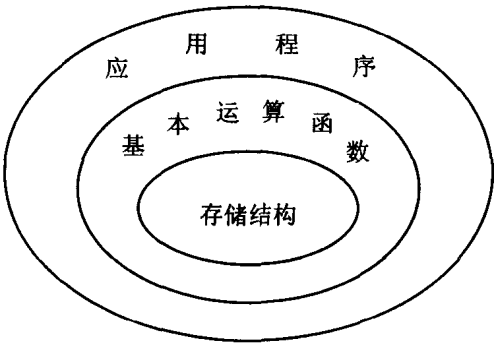


图 1.8 应用程序的结构

1.2 算法和算法分析

1.2.1 算法及其表示

算法是对特定问题求解步骤的一种描述，它是指令的有限序列，其中每条指令表示一个或多个操作。算法有以下5个重要特征。

① 有限性：一个算法必须总是（对任何合法的输入值）在执行有限步之后结束，且每一步都可在有限时间内完成。
② 确定性：算法中每一条指令必须有确切的含义，不会产生二义性。
③ 可行性：一个算法是能执行的，即算法中描述的操作都是可以通过已经实现的基本运算的有限次执行来实现。
④ 输入性：一个算法有一个或多个输入。
⑤ 输出性：一个算法有一个或多个输出。

描述算法的方法很多，有的采用类PASCAL语言，有的采用自然语言。本书采用C/C++语言来描述算法的实现过程。

【例1.2】 有下列两段描述：

```
(1) void exam1()
    {
        n=2;
        while (n%2==0)
            n=n+2;
        cout << n << endl;
    }
```

```
(2) void exam2()
    {
        y=0;
        x=5/y;
        cout << x << "," << y << endl;
    }
```

这两段描述均不能满足算法的特征，试问它们违反了算法的哪些特征？

解：（1）是一个死循环，违反了算法的有限性特征。（2）出现除零错误，违反了算法的可行性特征。

1.2.2 算法分析

算法分析的两个主要方面是分析算法的时间复杂度和空间复杂度，其目的不是分析算法是否正确或是否容易阅读，主要是考察算法的时间和空间效率，以求改进算法或对不同的算法进行比较。一般情况下，鉴于运算空间（内存）较为充足，所以把算法的时间复杂度作为分析的重点。

算法的执行时间主要与问题规模有关。问题规模是一个和输入有关的量，例如，数组的元素个数、矩阵的阶数等。所谓一个语句的频度，即指该语句在算法中被重复执行的次数。算法中所有语句的频度之和记做$T(n)$，它是该算法所求解问题规模n的函数。当问题的规模n趋向无穷大时，$T(n)$的数量级称为渐进时间复杂度，简称为时间复杂度，记作$T(n)=O(f(n))$。

上述表达式中“O”的含义是$T(n)$的数量级，其严格的数学定义是：$T(n)$的数量级表示为$O(f(n))$，是指存在常量$C\neq 0$和n_0，使得$\lim\limits_{n\to\infty}\dfrac{|T(n)|}{|f(n)|}=C\neq 0$成立。

另外，由于算法的时间复杂度主要分析$T(n)$的数量级，而算法中基本运算的频度与$T(n)$同数量级，所以通常采用算法中基本运算的频度来分析算法的时间复杂度，被视为算法基本运算的一般是最深层循环内的语句。

用数量级形式$O(f(n))$表示算法执行时间$T(n)$的时候，函数$f(n)$通常取较简单的形式，如1、$\log_2 n$、n、$n\log_2 n$、n^2、n^3、2^n等。在n较大的情况下，常见的时间复杂度之间存在下列关系：

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

【例1.3】 分析以下算法的时间复杂度。

```
for (i=1;i<=n;i++)                                    //①
   for (j=1;j<=n;j++)                                 //②
   {
       c[i][j]=0;                                     //③
       for (k=1;k<=n;k++)                             //④
          c[i][j]=c[i][j]+a[i][k]*b[k][j];            //⑤
   }
```

解：这里采用两种方法分析算法的时间复杂度。

方法1：语句①的执行频度为$n+1$（$i\leqslant n$需执行$n+1$次）；语句②的执行频度为$n(n+1)$；语句③的执行频度为n^2；语句④的执行频度为$n^2(n+1)$；语句⑤的执行频度为n^3。

算法的执行时间是其中每条语句频度之和，故：

$$T(n)=2n^3+3n^2+2n+1=O(n^3)$$

方法2：上述算法中的基本运算是语句⑤，其执行频度为n^3。则：

$T(n)=n^3=O(n^3)$

从中看到，两种方法的结果相同，而方法2更加简洁。

【例1.4】 给出以下算法的时间复杂度。

```
void func(int n)
{
    int i=1,k=100;
    while (i<n)
    {
        k++;i+=2;
}
}
```

解： 设while循环语句执行的次数为m，i从1开始递增，最后取值为$1+2m$，有：

$$i=1+2m<n$$

即

$$m<(n-1)/2=O(n)$$

即该算法的时间复杂度为$O(n)$。

【例1.5】 给出以下算法的时间复杂度。

```
void sort(int j,int n,int a[])
{
    if (j==n)    //子数组只有一个元素                                    //①
        cout << a[n];
    else
    {
        for (i=j+1;i<=n;i++)                                            //②
            if (a[i]<a[j])
            {
                temp=a[i];a[i]=a[j];a[j]=temp;
            }
        cout << a[j] << " ";                                            //③
        sort(j+1,n,a);                                                  //④
    }
}
```

解： 从上述算法中看出：sort(j,n,a)用于对a[j]~a[n]元素进行排序。当$j=n$时，需要排序的部分只有一个元素，此时只需输出一个元素a[n]，时间为常量，用1表示；否则执行else子句，其中语句①需要执行$n-j$次，语句③是输出一个元素，时间为常量，用1表示，语句④是递归调用语句，用$T(j,n)$表示sort(j,n)的工作时间，语句④的工作时间就是$T(j+1,n)$，由于a[$j+1$]~a[n]的排序时间与a[j]~a[$n-1$]的排序时间相等，所以$T(j+1,n)=T(j,n-1)$，则有

$$T(j,n)=T(j+1,n)+(n-j+1)=T(j,n-1)+n-j+1$$

其中，1为语句③的工作时间，$n-j$为语句②的执行时间。若用$T(n)$表示sort(1,n)的执行时间，