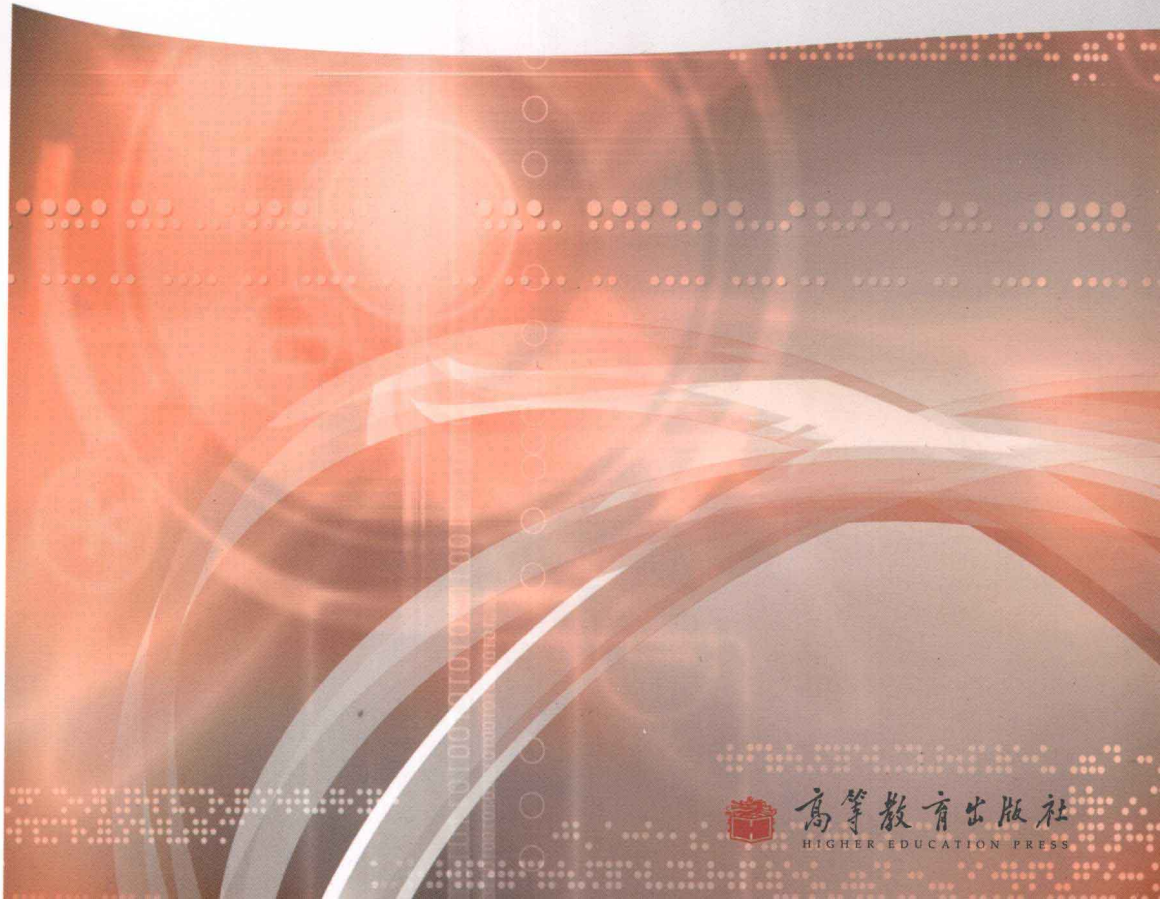
高 等 学 校 教 材

数据结构：炫动的0、1之弦

Data Structures: Dazzling String of 0 and 1

邹恒明 著



高等教育出版社
HIGHER EDUCATION PRESS

内容提要

本书从软件设计师和系统架构师的视角对数据结构进行阐述。通过两个角度的对望，以实际生活中的“问题”为驱动，以计算机软件设计师的“使用”为轴线，对每一种数据结构出现的动机、发展逻辑、表示方式、实现细节进行演绎，再现了数据结构的本质和内涵。本书讨论的结构包括栈、队列、表、栈表、索引表、跳转表、哈希表、二叉（查找）树、AVL 树、伸展树、B/B + 树、堆、幂堆、斐波那契堆、图、集合、划分和标准模板结构等。全书逻辑性强，注重阐述如何从一种想法转换为一种设计，又如何从设计转化为具体程序，从而化复杂为简单、化抽象为具体，大幅度降低学习和掌握数据结构的难度。为了方便准备考研的读者，本书还提供了2009—2010 年两年的全国硕士研究生入学统一考试中数据结构部分真题的详细解析。

本书可作为高等学校计算机科学与技术、软件工程等相关专业数据结构课程教材，也可供程序设计人员及参加全国硕士研究生入学统一考试的应试者参考使用。

图书在版编目（CIP）数据

数据结构：炫动的0、1之弦／邹恒明著. —北京：高等教育出版社，2012.1

ISBN 978-7-04-032486-0

Ⅰ.①数… Ⅱ.①邹… Ⅲ.①数据结构-研究生-教材 Ⅳ.①TP311.12

中国版本图书馆 CIP 数据核字（2012）第000008号

策划编辑 倪文慧	责任编辑 倪文慧	封面设计 王 洋	版式设计 马敬茹
插图绘制 杜晓丹	责任校对 胡晓琪	责任印制 韩 刚	

出版发行	高等教育出版社	咨询电话	400-810-0598
社 址	北京市西城区德外大街4号	网 址	http://www.hep.edu.cn
邮政编码	100120		http://www.hep.com.cn
印 刷	北京市密东印刷有限公司	网上订购	http://www.landraco.com
开 本	787×1092 1/16		http://www.landraco.com.cn
印 张	23.5	版 次	2012年1月第1版
字 数	570千字	印 次	2012年1月第1次印刷
购书热线	010-58581118	定 价	32.00元

物料号 32486-00

前言:软件设计师的数据结构

有这样一个故事,很久以前,一个财主叫仆人去买酒,仆人向财主要买酒的钱。财主回答说:“用钱买酒算不上本事,没有钱买来酒才算有本事。”仆人无言退出。过了一会儿,仆人回来,交给财主一个空酒瓶。财主生气地问:“酒呢?”仆人回答说,“从有酒的瓶子里喝出酒算不上本事,从空瓶子里喝到酒才算有本事”。财主无言以答。

当然,这个故事多半不是真实的,它只不过是穷人用来讽刺财主而已。但这个故事却道出了一个程序设计的奥秘:程序这种“酒”需要某种“钱”来买,或者程序这个“空瓶”需要装进“酒”才能喝。这个“钱”或“酒”是什么呢?不同人有不同的答案,但多数软件设计师会同意的答案是各种编程工具。程序设计需要使用精巧的工具来应对应用的复杂性。没有工具,设计优美的程序将是空话。而所有的工具里面,数据结构是极为关键的一个。尤其是在广泛使用面向对象程序设计的今天,所有的一切都围绕着数据而转移。数据需要以某种方式存放和访问,而存放和访问就不得不涉及数据结构。其实,从最根本的意义上看,数据的存放与访问就是数据结构(本书后续章节的数据结构定义将仔细阐述这点)。

1. 软件设计师的数据结构

作为软件设计师的必备工具,数据结构具有重要地位。没有数据结构,软件设计师就不能成为软件设计师,甚至程序本身也无法存在。但是在众多专业课程中,数据结构被很多学生认为是一门很难学习的课程。

2006 年 1 月,一名上海交通大学的学生在数据结构期末考试中交了白卷。严格地说,学生交的并不是白卷,而是一份“檄文”。在这份“檄文”中,该同学开门见山地说:“拿着试卷的那一刻,我怀着不安的心情写下了我的班级、姓名和学号。然而与以前考试所不同的是,我一个字都不会写,准确地说是一个字母也不会……”。

一名大学生在数据结构课程考试上一个字都答不上来,这不能不令人深思。也许读者会认为这位学生平时不认真学习,因而答不上来。但这位学生在其“檄文”的末尾写到:“附:本人不玩游戏,上课几乎每节都去,可真的是学不来啊……”。

看来这位“白卷英雄”不是因偷懒而交了白卷,而是觉得数据结构这门课程太难了!事实上,觉得这门课程难的并不只有这一位学生。在参加该次数据结构统考的十几个班里,有几个班的不及格率超过了 50%。答得最好的班也有近 15% 的学生不及格。因此,数据结构课程难学这个结论恐怕不是基于孤立的个案,而是一个普遍的现象。

2. 数据结构的思考

通过收集学生反馈并进行归纳与分析后发现,对于很多人来说,数据结构的概念并不难,学生对什么是栈、队列、树、图等通常都能够应对自如,真正的难点是:

(1) 如何实现从数据结构概念到程序实现的跨越(即如何实现一个数据结构);

(2) 如何实现从实际应用到数据结构抽象的跨越(即如何利用数据结构解决实际问题)。

数据结构的概念既抽象又具体,抽象是因为它可以脱离计算机或其他实际环境而存在;具体则是因为它可以在计算机中成为具体的程序代码。由于数据结构是抽象与具体的统一,理解与掌握它就要求能够跨越横亘在抽象与具体之间的鸿沟,并能够随意在其间往返。而这并不容易。

以队列结构的定义为例,队列结构是一个某类型数据项组成的有穷序列,它支持下述操作:

- 创建一个空队列;
- 如果队列未满,可以将一个元素加入到队列末尾;
- 如果队列未空,可以将队列头元素删除,也可以将队列头元素取出;
- ……

如果将这个抽象队列在计算机上实现,初学者经常会不知所措,如何在计算机上创建一个空队列?空队列是什么意思?队头在哪里?队尾又在哪里?显然,从上述抽象定义到队列在计算机中的具体实现之间存在一道鸿沟。这道鸿沟对于那些刚进入大学一年多的学生来说过于宽大,以至于难以跨越。类似地,一个学生也许很清楚抽象层次上图结构的逻辑定义和含义,可是具体在计算机中如何表示图,进而又如何对图进行各种操作或运算则显得有些迷茫。

数据结构的第二个难点是如何在具体问题上选择数据结构。例如,构建一个搜索引擎到底会用到哪些数据结构?或者说,应该使用何种数据结构才好?数组、矢量结构、栈、树、哈希表或者其他?这个问题涉及另外一种跨越,即从实际应用的具体到数据结构的抽象之间的跨越。如果说从抽象到具体已经很难,则从具体到抽象就更难了。而且,从实际应用到数据结构的这种具体到抽象的映射是多对多的:一个实际应用可以用多种数据结构实现,而一种数据结构可以用来解决多种实际应用。有时一个实际应用甚至可能需要同时使用多种数据结构,而构建一种有效的数据结构组合难度显然更大。

3. 本书的设计理念

本书是为解决上述数据结构的难点而作,即在跨越从数据结构概念到程序设计实现和从实际应用到数据结构抽象这两个鸿沟上下功夫。对于第一个鸿沟,即从抽象到具体的跨越问题,本书将从一个软件设计师的角度对数据结构进行讲解,因为从抽象到现实的映射是一个软件设计师的职责;而第二个鸿沟牵扯到实际应用到数据结构之间的映射,也就是具体到抽象的映射。本书的应对之道是从系统架构师的角度对数据结构进行讲解。因为从具体应用到抽象数据结构的映射是一个系统架构师的职责。而具体措施就是以实际问题为驱动来引入并讨论数据结构。

本书采取上分下合的策略,将数据结构的内容与算法进行适度的剥离(上分,即与上面的算法知识适度分离),与程序设计更加靠近(下合,即与下面的程序设计更加靠拢)。本书对算法的讨论仅限于与数据结构联系紧密的查找与排序操作,而对算法的设计与分析几乎全部略去,只在确实需要的地方提供感性的算法分析,即大概论述一下某个数据结构的操作可能时间较长或所需空间较大,而不对它们进行精确分析。这样就可以在避免详细论述算法的情况下对数据结构之间的优劣进行比较,能够集中精力讨论数据结构,而不是将时间在数据结构和算法之间来回分

配,从而提高讨论的效果。

事实上,本书是从程序设计的角度来讨论数据结构的。在设计任何一种数据结构时都会考虑到编程的方便性和实用性,数据结构在很大程度上是为了程序设计而存在的,程序设计中的很多概念和构造,如析构器(destructor)、拷贝构造器(copy constructor)等,均是为了要正确操控数据结构才发明的。本书认为,围绕程序设计来讨论数据结构和围绕数据结构来讨论程序设计,是跨越数据结构抽象与具体之间鸿沟的根本手段。

4. 本书的写作目的

本书试图从新的角度阐述和分析数据结构,以一种更为容易接受和理解的方式。通过与算法的适度分离和与程序设计的更加靠近,来将数据结构的本质、内涵、程序设计实现和具体应用植入读者的头脑中,以使读者能够使用各种数据结构来编写出真实有用的程序和软件。具体来说,本书欲达到的目的包括如下一些方面:

- 训练读者的数据结构思维,使其认识到数据结构的本质和内涵;
- 介绍结构化问题解决技术和数据抽象原则;
- 从系统架构师和软件设计师两个角度跨越具体与抽象之间的鸿沟;
- 帮助读者发觉精巧数据结构给程序所带来的巨大改善;
- 帮助读者构建自己的数据结构工具库;
- 介绍如何概括性地评价一个数据结构和程序的成本;
- 介绍如何应用数据结构来解决实际问题。

本书使用的程序设计语言是C++。选择C++的主要原因是其具有普遍适用性、适度结构性和较大灵活性。不同的程序设计语言在构造不同的数据结构时其难度与复杂性不尽相同。而在C++上构建各种数据结构不会遇到过于困难或复杂的情况。

5. 本书的组织方式

本书从内容上分为6部分,包括结构基础篇、线性结构篇、数据结构上的通用操作篇、非线性结构篇、扩展结构篇和附录篇。这里,我们将数据结构分为三大类:线性结构,非线性结构和扩展数据结构。这种分类依赖的是数据结构本身的异同,而不是操作技术。

(1) 结构基础篇为整本书的讨论奠定基础,讨论数据结构的定义及相关概念。该篇仅包括数据结构基础一章。

(2) 线性结构篇讨论数据结构中最直观的线性结构。这里的线性指的是元素之间的相对关系为线性,即每一个元素只有前后两个邻居,首元素没有前邻居,尾元素没有后邻居(注意这与逻辑上的线性概念有所不同)。该篇共包括栈结构、队列结构和表结构3章。

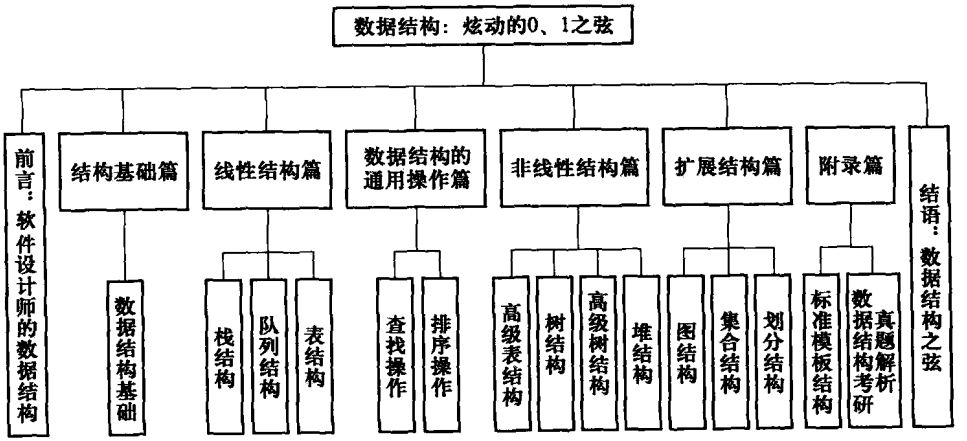
(3) 数据结构上的通用操作篇讨论与数据结构联系紧密的两种操作:查找和排序。这部分内容属于算法与数据结构的共同内容。从一个方面讲,查找与排序可以看做是表结构的两种应用或者两个操作;从另一个方面看,它们确实涉及算法的思想。本书着重对查找与排序的实际实现进行讨论,并涉及不同数据结构对查找与排序算法的影响(反之亦然)。本篇共包括查找操作与排序操作两章。

(4) 非线性结构篇讨论数据结构中的非一维结构。该篇共包括高级表结构、树结构、高级树结构和堆结构4章。

(5) 扩展结构篇讨论数据结构里面构建于基本线性和非线性结构之上的结构。该篇一共包括图结构、集合结构和划分结构3章。

(6) 附录篇包括标准模板结构和考研真题解析两部分内容。考虑到目前编程环境的演变,越来越多的人都会不同程度地遇到函数和结构库,学习和掌握这些标准函数和结构的使用也成为一个程序员的重要技能,本书用附录一来专门讨论标准模板数据结构。另外,为了方便准备考研的读者,附录二提供了2009—2010两年的全国硕士研究生入学统一考试中数据结构部分真题的解析。

本书的内容组织如下图所示。



6. 本书特色

本书具有如下几个特色:

(1) 从软件设计师和系统架构师两个角度来展开对数据结构的讨论。通过两个角度的对望,以"问题"为驱动,以"使用"为轴线,对每一种数据结构出现的动机、发展逻辑和表示方式进行演绎,玲珑剔透地再现其内在的本质和内涵。

(2) 注重阐述如何从一种想法转换为一种设计,又如何从设计转化为具体程序,从而化复杂为简单、化抽象为具体,帮助读者跨越横亘在高度抽象与高度具体之间的鸿沟。

(3) 以实例示范,给出了所有数据结构的代码实现。所有代码均在微软 Visual Studio 2008 环境下编译通过并执行。本书提供的代码实现除包括通常的栈、队列、线性表、链表、二叉树、AVL树、二叉堆之外,还包括不常见的跳转表、B+树、伸展树、幂堆、集合结构、划分结构等。

(4) 去繁就简,将注意力集中在数据结构本身的设计和构造上,将与数据结构本身无关的外在因素抛开。例如,本书使用的语言虽然是C++,但其中并未涉及C++语言的对象层次架构,代码设计也以简单为宜(例如,放弃了模板编程风格);对各种复杂的语言构造也敬而远之。本书认为,在数据结构的实现中使用复杂或精巧的程序构造危害甚大,它将导致读者将大部分时间

花在理解程序语言的构造上,而丧失对数据结构本身的关注;初学者也可能将其与数据结构本身混淆起来,以为它们是数据结构实现的(不可缺少的)一部分,从而严重干扰他们对数据结构的理解。

7. 本书使用方式

本书可供2至4个学分的课程教学使用。教师可根据课程学分调节所覆盖的内容。如果是4个学分的课程,建议覆盖本书全部章节。如果是3个学分的课程,则可跳过第8章的二叉查找树的高度,第10章的幂堆、斐波那契堆,第12章集合结构和第13章划分结构等内容。对于2个学分的课程,则在3个学分课程的基础上,再跳过第7章,第9章的B树、B+树、伸展树、树和森林和第11章的图的拓扑排序操作等内容。

8. 致谢

本书得到了浙江大学计算机学院陈越教授认真细致的审阅,高等教育出版社的有关人员也为本书的出版做了大量的审读工作,在此,特向她们表示由衷的谢意。由于作者水平和认识所限,本书难免存在疏漏之处,还请读者批评指正。

现在就让我们翻开本书,以软件设计师的视角,在抽象与具体的沟壑上穿越,经受从未经历的简约和雅韵,在以前不曾达到的境界上察看和把握数据结构的精髓吧。

邹恒明

2011年10月于莘庄

目　录

第1章　数据结构基础

没有结构不能成方圆

在远古时期，世界上并没有什么国家，人类是按族群聚居在一起的。那时的族群规模很小，除了一个族长或酋长外，其他人都是平等的，就是族长自己也是整个族群中的平等一员。但随着族群人数的增加，人类发现这种平等结构已经不足以维持整个族群的“和谐”运作，于是氏族社会逐渐形成，慢慢地发展壮大直到形成国家。在一个国家，仅靠一个领导人来管理显然不够，于是就形成了形态各异的国家管理结构。

既然管理人需要不同的社会群体或机构（氏族、公社、国家等），管理大量数据是否也需要数据结构呢？

1.1　什么是数据结构

顾名思义，数据结构指的是某种关于数据的结构，或者由数据组成的一种结构，是数据之间关系的一种逻辑描述。

例如，将一个班级所有学生的名单按姓氏笔画从少到多排成一个序列。这个序列就是一个数据结构，它表明的是一个班级中所有学生之间的线性序列关系，如图1-1所示。

王一
左二
右三
李四
张五
赵六
秦七
蔡八

图1-1　按学生姓氏笔画顺序排列的数据结构

不过，上面给出的数据结构定义并不完整。因为将数据按某种方式组织起来的目的是为了对它们进行进一步的操作。如果组织数据之后并无任何操作，则这种组织毫无意义。例如，按图1-1将学生姓名排列起来就是为了查找方便。

因此，数据结构除了表示数据之间的关系外，还包括对这些数据进行的操作。这些操作必须能够保持数据结构的属性不发生变化。例如，对图1-1的表进行查询并不会改变该表的线性序列属性。因此，归纳起来可以得出数据结构扮演的两个角色是：数据封装和操作支持。

1.2　数据结构的定义

根据上面的陈述，可以得到数据结构的一个较为完整的定义：数据结构就是数据之间关系的

一种逻辑描述及在这些数据上可以进行的操作集合。该定义是从逻辑层面说的,因此可称为抽象数据结构。

例如,可以将一个队列定义为一个一维空间的记录序列,删除在队列的一端进行,插入在队列的另一端进行。这里应注意到该队列定义与任何物理现实无关,因而是抽象的。

也有人把数据结构定义为:一种在计算机中有效存放和组织数据的方式(可参见 Wikipedia 对数据结构的定义)。本书并不认同该定义。首先,该定义覆盖的是一种较为具体的数据存放方式,即在计算机中存放数据,这就脱离了抽象的数据结构;其次,一个结构并不只是效率高时才能存放数据。低效率的存放方式一样可以存放数据,虽然效率低,但它还是数据结构。因此,该定义并不是很恰当。

从计算机的角度而言,数据结构可以定义为:数据结构是数据之间关系在计算机中的表示,以及一组对该数据允许的操作。该定义是从物理层面上说的。因此,可称为具体数据结构。

例如,也可以如此定义队列:它是一片连续的一维内存空间,该空间有两个指针,分别指向其中存放的首端记录和末端记录,删除操作对首端指针所指向的记录进行,每删除一条记录将首端指针增加1;而插入操作在末端指针所指向的位置的下一个位置进行,每插入一条记录则将末端指针增加1。由于该定义是在一个物理现实上进行,因而是具体的。

对于一个程序设计员来说,所关心的数据结构通常是具体数据结构;而对于一个系统架构师或系统设计员来说,所关心的则通常是抽象数据结构。可以说,具体数据结构是抽象数据结构或抽象数据类型的物理实现。

为了避免混淆,本书不采用抽象数据结构和具体数据结构的称谓,而将抽象数据结构称为数据结构;具体数据结构称为数据结构的实现。

1.3 数据结构的目的

细心的读者可能已经感觉出,设计数据结构的目的是为了让计算机能够对数据进行更好的处理。但到底什么是更好的处理呢?当然不同的人对"更好"有着不同的理解。

本书认为,从软件设计师的角度来看,"更好"包括两个方面:

(1) 概念上一致;

(2) 更高的处理效率。

概念上一致指的是将数据归类为整齐划一的表示。例如,可以用姓名来表示一个学生,这样所有的学生都归结为一个名字,从而将学生的表示和计算机之间的联系统一为一个名字。也可以更进一步,将学生的姓名、地址、性别、专业等信息归结为一个编号。这样,不管一个学生的姓名是什么,有多少个字符,或者用字多么生僻,对于计算机来说,都是一个编号而已。这样就形成了概念上的一致性。这种一致既便于人的理解,也便于数据的组织和计算机的处理。而用于表示一致性的数据结构就是所谓的聚类结构,如众所周知的结构(struct)和类(class)。这部分内容

由程序设计课程覆盖，本书不再赘述。

更高的处理效率则是指提高计算机处理数据的效率。例如，如果将一个大学的所有学生姓名随意排放，则查找一个特定的学生将非常费时费力。如果把这些学生记录按照学号的递增序或递减序组织成一种线性结构，就可以提高查找的效率。为了进一步提高查找、插入、删除一个学生记录的速度，还可以把它们按专业、年级和班级组织成层次结构，从而进一步提高执行基本操作的效率，达到降低时间代价的目的。

而用于提高处理效率的数据结构就是本书要讨论的范畴。它们包括栈、队列、线性表、链表、树、堆、图、集合、字典等。不同的数据结构适用于不同的应用，有的数据结构就是专门为某种应用而设计的。例如，B 树就特别适用于数据库和文件系统，而哈希表则常常在编译器里面使用，字典用来支持任何需要在海量数据里面进行搜索的应用，等等。

1.4 数据结构的种类

不管是时间关系，还是空间关系，数据之间的关系并不只有一种。例如，在空间关系上，可以有上下左右及同一位置等关系，而时间关系则有前后或相同的关系。如果数据项很多，它们之间在空间或时间上可以形成的关系几乎是无限的。而既然数据结构是数据之间关系在计算机中的表示，那就意味着我们设计的数据结构需要能够表示这无数种可能的关系。

显然，研究无穷的东西非常困难，那应该怎么办呢？

答案是抽象。将无限多的关系种类里面的非关键属性去除，只取其中的共性来设计数据结构。例如，对于数据项 A、B、C 来说，它们之间的关系可以是 A 在 B 的前面，B 在 C 的前面；或者 C 在 B 的前面，B 在 A 的前面。显然，对于个体的数据项来说，这两种结构是不同的，但如果抛开个体数据项将会发现，这两种结构实际上是一种类型：线性结构。

可以发现，抛开非关键属性后，数据之间的关系种类就非常有限了。事实上，从逻辑上看，它们之间只有线性和非线性关系。线性关系就是数据之间可以排列成一条直线，除首尾两个数据项外，其他数据项都只有一个直接前驱数据项和一个直接后继数据项。

非线性关系就是不符合上述规则的关系。由于不符合上述规则的关系范围广泛，为了研究方便，将其进一步归类为层次结构和网状结构。层次结构下，数据分为不同的层来存放，一层数据只与相邻的层有直接联系。层次结构的常见形式是树形结构，在此种结构下，一个节点可以对应多个节点，它们之间的关系是一对多的。而网状结构则不存在这样的层次关系，常见的网状结构有图结构。在此种结构下，节点之间的关系是多对多的。

此外，还有一种特殊的关系：数据之间没有任何关系。这种情况下，如果一组数据要归类的话，就形成所谓的集合。虽然如此，但在具体实现这种没有关系的数据集合时，落到计算机上的必然还会是某种结构，如线性结构或非线性结构，即可以将其看做是构建在其他结构上的一种结构。正是从这种意义上来看，这种结构也称为扩展数据结构。

除了上面的线性、非线性和扩展数据结构分类外，还可以从其他标准来对数据结构进行分类，这些分类方式考虑的不是数据项之间的关系，而是数据结构本身作为一个整体时的某些属性。这些分类带来的益处见仁见智，本书认为，了解它们能使读者增强对数据结构的认识。具体来说，可以从以下5个方面来进行分类：

（1）从数据结构存放的介质来看，可以划分为外存数据结构和内存数据结构。

（2）从数据结构维持方来看，可以划分为被动数据结构和主动数据结构。

（3）从数据结构存储的时间来看，可以划分为持久数据结构和即时数据结构。

（4）从数据结构整体与部分的关系来看，可以划分为递归数据结构与非递归数据结构。

（5）从数据结构与历史的关系来看，可以划分为历史独立数据结构与历史痕迹数据结构。

外存和内存数据结构

外存指的是计算机外部存储介质，通常就是磁盘。注意这里的计算机指的是CPU主板。磁盘在主板之外，因此称之为外存。当然，磁带、光盘等也是外存。而在CPU主板上的存储器则称之为内存。由于内、外存的访问速度差异很大，在构造数据结构时的考虑因素就不尽相同。有的数据结构用在内存比较合适，而有的用在外存比较合适（如B树）。

被动和主动数据结构

被动数据结构指的是内容只能由数据结构之外的线程或进程进行改变的数据结构。绝大部分数据结构都属于此种类型。例如，一个名字序列、一张表，它们本身只存放数据，对数据或结构进行修改需要独立于这些序列或表格之外的程序来进行。主动数据结构指的是本身带有线程和进程的数据结构。这些进程或线程通过某种操作给用户呈现出一种被动数据结构所无法实现的抽象。例如，一个自管理队列就是一个主动数据结构。从抽象上看，一个队列应该是没有长度限制的，而在计算机中实现一个队列则有着物理限制。为了使队列的逻辑抽象保持稳定，需要使用自管理队列。这个自管理队列维持着两个物理队列：一个在内存；另一个在磁盘上。当内存的队列满了后，自管理队列里面的线程或进程将把后续的数据放在磁盘上；等内存的队列有空间时，再将磁盘上数据往内存传送，以此来维持无限长度队列的抽象。

持久与即时数据结构

持久的数据结构就是数据结构的老版本皆得到保留的数据结构。这样在需要时可以查询到老版本的数据。例如，ZFS文件系统里使用的就是持久数据结构，当一个文件更新时，其老版本仍然保留。即时数据结构则只保留最新的版本，任何老版本信息都被消除。

递归与非递归数据结构

如果一个数据结构是由规模更小、但结构相同的子结构构成，则称为递归数据结构。树是一个典型的递归结构。如果一个数据结构的组成部分的构造与整体构造并不相同，则为非递归数据结构。

历史独立与历史痕迹数据结构

数据结构是用来存放数据的，而存放的特性就是记忆，即记住数据项的状态或者取值。随着各种操作的进行，数据结构的内容在发生变化，而这一系列变化就形成了该数据结构的历史。如

果一个数据结构能够让用户从其当前状态判断过去发生的事情，也就是推断历史，则该数据结构就是历史痕迹数据结构；如果无法从当前的内容或状态判断过去的内容或状态，则这样的数据结构就称为历史独立数据结构。显然，有时需要历史痕迹数据结构，有时则需要历史独立数据结构。需要历史痕迹数据结构的理由很简单：用户需要知道过去发生的事情，以便查账，或者在发生问题时可以进行有益的调查。而需要历史独立数据结构的理由也并不复杂：用户不希望过去的历史被人知晓，或历史对用户没有价值。

历史独立数据结构和历史痕迹数据结构不属于基本的数据结构讨论范围，本书将不予讨论。有兴趣的读者可自行参考相关资料。

1.5 数据结构与抽象数据类型

前面已经说过，数据结构是一种抽象，它将数据的个体属性去除，只考虑数据项之间的关系。既然是一种抽象，有读者自然会联想到程序设计课程中讲过的抽象数据类型。那么数据结构与抽象数据类型之间有什么关系吗？

先来比较一下抽象数据类型与数据结构的定义。抽象数据类型是一组数据值或对象的集合及可以对这些数据值或对象进行的操作。它识别一组数据值或对象的共同特征，这些特征把该组数据值或对象辨认为同一种类型。数据结构则是数据之间关系的逻辑表述及其可以进行的操作。二者之间看上去似乎是很相似的。

抽象数据类型与数据结构确实关系紧密。事实上，从某种意义上说，甚至可以将抽象数据类型看做是数据结构，而将数据结构看做是抽象数据类型。只不过在程序设计里面提到的抽象数据类型大部分是所谓的原子数据类型，即它的数据值不可以再进行分解。如数据类型 int。这种原子数据类型看上去与数据结构似乎不同，因为不存在所谓的数据元素之间关系一说。但如果数据类型为聚合类型(aggregate type)，则情况就有所不同了。聚合数据类型的数据值可以进一步被分解为组成元素。这些数据元素之间通常具有某种结构，或者换句话说，它们之间具有某种关系，这就与本书定义的数据结构连通起来了。

因此，可以说数据结构是一种数据类型，它的数据值可以进一步分解为数据元素，该数据元素可以是原子数据值，也可以是另外一种数据结构，数据元素之间有一个关系。例如，C ++ 或 Java 里面的类(class)就是一个标准的数据结构。从另外一个角度看，数据项既有逻辑表示也有物理表示。逻辑表示就是抽象数据类型里的数据项定义，如整数；物理表示则是数据结构中的数据项实现，如 16 位整数。数据结构与抽象数据类型的关系如图 1-2 所示。

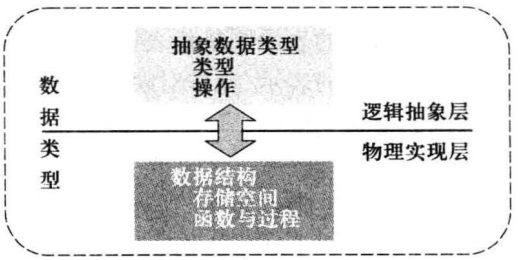


图 1-2 数据结构与抽象数据类型的关系

1.6 数据结构的特性

根据上述对数据结构的讨论，可以发现，数据结构具有如下特性：

- 精确：哪些值属于该数据结构，哪些不属于，一清二楚。
- 独立：它独立于任何具体的实现，我们关心的是逻辑上的数据之间的关系。
- 封装：实现的细节被完全封装（考虑类）。

数据、元素与记录

数据结构是为数据而设计的。在计算机界，数据则是对电子信息的一种统称。能够单独作为一个整体进行操作的数据通常称为数据项，有时也称为记录，而不能分割的数据项就称为数据元素，或简称为元素。由于数据结构本身只是一种架构，并不在意存放在其中的具体数据到底是何种形态（是不可分割的元素还是由多个元素组成的记录或数据项），在本书对数据结构的讨论中，会交替使用元素、数据项、数据记录等术语。

时间关系与空间关系

根据本书的定义，数据结构捕捉了数据之间的关系。但是数据之间的关系指的是何种关系？是数据项之间的空间位置关系，还是数据项之间的时间关系？实际上，空间关系和时间关系都可以在数据结构中得到表示。例如，图1-1的学生名单可以表示一个空间关系，它代表的是全班学生列队时的前后站立顺序：即学生王一在列队时站在全班最前面，左二站在第二个位置，以此类推。当然，图1-1也可以表示一个时间关系，即学生王一是全班第一个注册某门课程的学生，左二为第二个注册该门课程的学生，以此类推。

一般来说，数据结构所表示的数据之间的关系既可以是空间位置关系，也可以是时间轴线关系，当然也可以是任何其他人为制造的某种序关系（例如大小关系）。但到底是哪种关系通常并不影响我们对数据结构本身的研究，因此本书后面在阐述数据结构时将不再在时间或空间或其他关系上进行说明，而是集中精力讨论每个数据结构本身的优劣。

状态、表示和实现

一个数据结构的状态是由其抽象数据结构所指明的该数据结构的当前取值或内容，一个数据结构给定状态的表示则是代表该数据结构状态的物理内存的实际内容，一个数据结构的实现则是从一个合法的表示/操作对映射到一个新的表示。

数据结构的信息冗余

一个数据结构除了表示需要的数据外，有时会包括一些非必需的额外信息。这些信息的提供还是为了提高处理的效率。例如，对于一个数据序列来说，除了表示这个序列的所有数据外，可能还会为这样一个结构增加一个长度的信息。这样如果想知道一个序列有多长，直接查询这个长度值即可，而不需要逐个数据项地来计数。