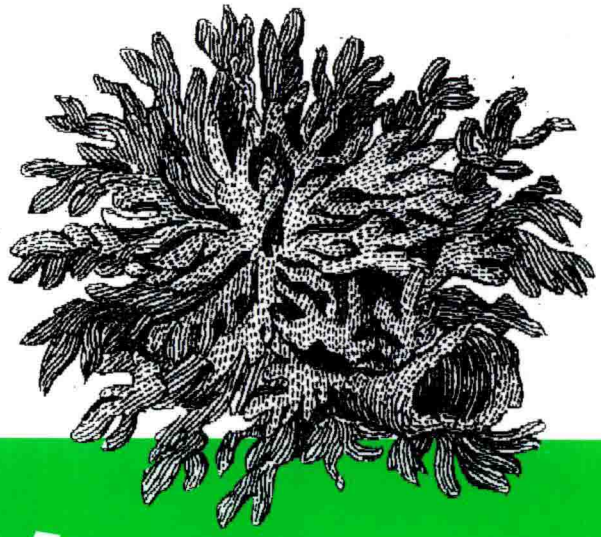
21 世纪高等教育
计算机规划教材



数据结构
C 语言描述

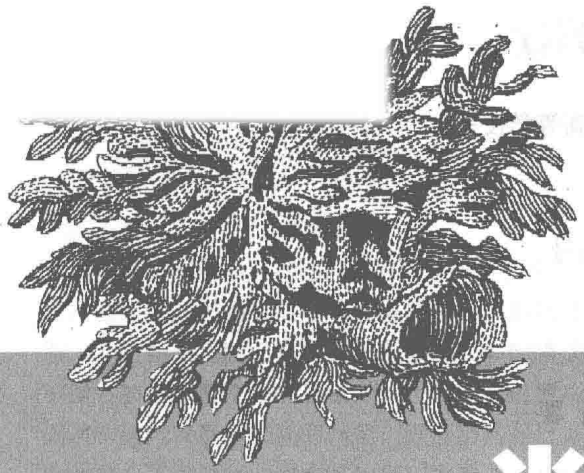
慕课版

◆ 张同珍 编著

+ **互联网 + 教材：** 人邮学院慕课平台作支撑，由上海交通大学张同珍副教授录制全套慕课
+ **注重思维方式的引导：** 采用了逻辑结构 + 物理结构 + 基本操作实现 + 典型应用的讲解模式
+ **C 语言代码实现：** 给出各种结构存储、操作算法的 C 语言实现代码，使抽象晦涩的算法转变为实用工具

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

21世纪高等教育
计算机规划教材



数据结构
C语言描述

慕课版

◆ 张同珍 编著

人民邮电出版社
北京

图书在版编目（CIP）数据

数据结构 : C语言描述 : 慕课版 / 张同珍编著. -- 北京 : 人民邮电出版社, 2018.8
21世纪高等教育计算机规划教材
ISBN 978-7-115-47603-6

Ⅰ. ①数… Ⅱ. ①张… Ⅲ. ①数据结构－高等学校－教材②C语言－程序设计－高等学校－教材 Ⅳ. ①TP311.12②TP312.8

中国版本图书馆CIP数据核字(2018)第001059号

内 容 提 要

数据结构是计算机及相关专业的基础课程。它不仅具有很强的理论性，也具有很强的实践性。本书对查找、排序进行了分析讨论，对线性结构、树结构、图结构采用了统一的讲解模式：逻辑结构+物理结构+基本操作实现+典型应用，并围绕这4个方面进行了详细讨论，条理清晰。另外，本书除了对各部分的操作实现算法进行理论分析之外，还用C语言进行了具体实现，从基本理论和基本技能两个方面对学生进行训练。

本书内容丰富、条理清晰、深入浅出、讲解详尽，适合计算机类、信息类、电类、自动控制类、数学类专业的学生使用，也适合软件设计人员、工程技术人员参考。

◆ 编　　著　张同珍
责任编辑　税梦玲
责任印制　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
固安县铭成印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：15　　2018年8月第1版
字数：397千字　　2018年8月河北第1次印刷

定价：45.00元

读者服务热线：(010)81055256　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字20170147号

前言
Foreword

通过对程序设计课程的学习，学生初步掌握了程序设计思想与方法，能够解决一些日常生活、学习、生产实践中遇到的数值和非数值性问题。但现实中遇到的问题往往涉及的数据量大且数据间关系纷繁复杂，学生会感觉到已学的知识不够用。如何将这些问题中的数据及数据间关系抽象出来并在内存中进行存储？如何在所选择的存储方式下用计算机解决各类问题？以及这些问题解决方法的优劣评判，将是数据结构这门课程要解决的问题。数据结构是计算机专业的核心专业基础课程，也是整个电类、信息类专业的核心基础课程，许多其他理工类专业也将它作为必修课程之一。绝大多数高校在计算机及相关专业硕士研究生入学考试专业课科目设置中，也将数据结构列为必考科目。

本书首先根据数据之间关系的不同，由简到繁地把数据结构分为线性、树和图 3 个部分。线性关系除了常规的线性表，还包括线性关系的两个特例——栈和队列，由于栈和队列是最常用的工具，因此对线性关系细分出线性表、栈和队列两章内容，树和图各成一章。对存储数据的查询和检索是对数据最频繁的应用，而排序又能提高对数据的检索效率，因此又加入了查找、排序两章内容。这样就组成了本书完整的七章内容。这七章内容在安排上遵循了计算机专业数据结构课程的教学大纲要求，同时兼顾了硕士研究生入学考试的要求。

同时，本书在编排结构上由浅入深，从问题的引入、概念的介绍、数据的描述、算法的设计实现、应用问题的解决，逐次递进，便于入门。课后配套的习题供学生进一步理解、巩固所学知识，题目后的“*”表示题目的难易程度，“*”越多表示该题目难度越大，无“*”表示该题目较简单。重要的概念、算法和易错点都配有慕课视频。下面对慕课视频的学习方法做出说明。

1. 购买本书后，刮开粘贴在封底的刮刮卡，获取激活码（见图 1）。

2. 登录人邮学院网站（www.rymooc.com），或扫描封面上的二维码，使用手机号码完成网站注册（见图 2）。

图 1　激活码

图 2　注册人邮学院网站

3. 注册完成后，返回网站首页，单击页面右上角的“学习卡”选项（见图 3），进入“学习卡”页面（见图 4），输入激活码，即可获得课程的学习权限。

图 3　单击“学习卡”选项

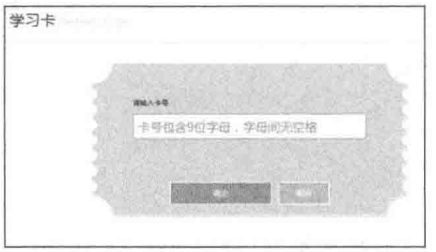


图 4　在“学习卡”页面输入激活码

4. 获取权限后，可随时随地使用计算机、平板电脑以及手机，根据自身情况，在课时列表（见图 5）中选择课时进行学习。

图 5　课时列表

人邮学院平台的使用问题，可咨询在线客服，或致电 010-81055236。

“数据结构”课程内容多、算法抽象，作者结合自身多年在上海交通大学讲授数据结构课程的教学经验，总结出了一些学习方法：既有理论的指导方法，也有具体操作上的指导方法。本书在内容讲解和章节小结中都会将这些方法适时提出并加以强调，希望对学生有所帮助。

全书在算法描述上使用了 C 语言（使用 C 语言的原因是它只使用了面向过程的程序设计方法，适合没有学过或不需要使用面向对象设计方法的学生），不仅有算法实现的 C 语言程序，还有对算法实现的测试代码，所有程序都在 Code::Blocks 10.05 下调试通过，学生可以直接运行，通过直观的运行结果方便地获得对算法的感性认识，更有利于理解抽象的概念和算法。

鉴于编者水平有限，书中难免有不足和疏漏之处，请各位读者批评指正，便于进一步改进。另外，与本书配套的 PPT 等资源，请前往人邮教育社区（www.ryjiaoyu.com）下载。

编　者

2018 年 6 月

目录
Contents

Chapter 01

第1章

绪论

■ 数据是外界信息进入计算机，并被计算机程序处理的符号。数据元素是数据的基本单位，如有一组存储在内存中的整数，这组整数是数据，数据中的每一个整数是数据元素；再如内存中有一组学生信息，每个学生信息是一个结构类型数据，包含了学号、姓名、年龄等字段，那么这组学生信息是数据，每一个学生的信息是数据元素。通常数据元素简称为元素。

1.1 数据结构的定义


数据结构定义

数据结构是指相互之间存在一种或多种特定关系的数据元素的集合，数据结构这门课程的研究对象是相同类型的一组元素之间的关系和关系操作（逻辑结构）、元素和关系在内存中的存储（物理结构）、在各种存储方式下关系操作的实现以及每种结构的典型应用。

1.1.1 数据的逻辑结构


逻辑结构

研究数据的逻辑结构，就是研究类型相同的一组元素和元素间的关系，元素关系可以分为以下几种。

（1）集合关系：元素间呈松散关系，结构中不同元素除了同属于一个集合，相互间无其他制约关系。如同班级里同学间的关系。

（2）线性关系：元素间呈现你先我后的顺序，是一种一对一的关系。如队列中元素间的关系，除了队首，每个元素有一个唯一的直接前驱元素；除了队尾，每个元素有一个唯一的直接后继元素。

（3）树形关系：元素间呈现一对多的关系。如家谱中人物间关系，一个人可以有多个儿子，却只能有一个父亲。

（4）图关系：元素间呈现多对多的关系。如城市间通过飞机航线形成的关系，如上海、北京、西安 3 个城市中，任何两个城市间都可以有直飞航线。

上述 4 种关系如图 1-1 所示。

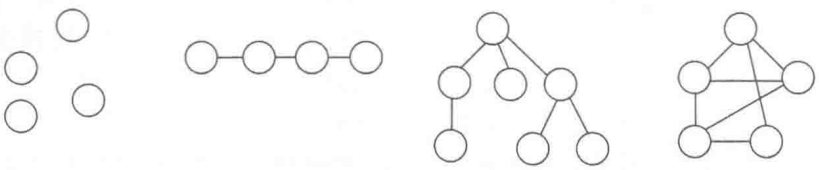


图 1-1 各种元素关系

线性关系、树形关系、图关系相应地使用线性结构、树形结构和图结构来解决，集合关系因其元素关系极为松散，可以结合应用其他几种结构，尤其线性结构，代为表示和处理。

逻辑结构通常可以用二元组描述为 Data_Struct=(D,R)，其中 D 是元素的集合，R 是关系的集合。如由整数 1～10 组成的有序集就是一个线性结构，描述表达式为

$D=\{x|1\leqslant x\leqslant 10,\ x\in \mathrm{N}\}$，$R=\{\langle x_1,x_2\rangle \mid x_1\in D,x_2\in D\}$。其中 $\langle x_1,x_2\rangle$ 表示一个有序偶，即 x_1 和 x_2 有顺序关系，x_1 是前驱，x_2 是后继。

1.1.2 基本操作


基本操作

基本操作（或称关系操作）是和数据的逻辑结构紧密相关的，它来源于关系自身的特点。无论哪种结构，基本操作都可以分为构造类、属性类、数据操纵类、遍历类和典型应用类。

（1）构造类：在内存中建立这种数据结构。如一个空的队列，有存储空间，无或有若干元素。

（2）属性类：对元素及元素之间的关系的各类查询。属于“东瞧瞧、西看看”，不影响元素及元素关系本身。如在线性结构中查询值为 X 的元素是否存在。

（3）数据操纵类：对元素或元素关系有改变的操作，如插入或删除某个元素，修改可以视作在同一位置上删除一个旧元素后再插入一个新元素。

（4）遍历类：对结构中的每个元素访问且只访问一遍。因其重要且有时又较复杂，常常是其他类操作的基础，如遍历图中的顶点元素，所以特意从属性类中分离出来。

（5）典型应用类：该种结构独特的应用，不同结构的典型应用各不相同。如线性结构可以解决队列问题，图结构可以解决两个城市间最短路径问题。

1.1.3 抽象数据类型

数据的逻辑结构连同基本操作组成抽象数据类型 ADT（Abstract Data Type）。抽象数据类型只和数据自身的逻辑特征相关。ADT 又区别于具体某种类型，数据结构只关心元素关系和关系操作，并不实际限定数据的具体类型，只要求元素的类型一致。

ADT 的描述可以用自然语言，也可以用伪代码，其内容包含元素集合、元素关系集合、基本操作。基本操作表明了现实生活中的一个个基本的问题，它给出了明确的已知条件和结果要求，具体实现依赖于数据和数据的关系在内存中如何存储，在逻辑结构分析阶段因不知道存储方式而无法给出 ADT 的描述。

1.1.4 数据的存储结构


存储结构

数据的存储结构也称物理结构，是数据结构在内存中的表示形式。数据要得到处理，首先必须进入内存，这里不仅要存储元素，还要存储元素间的关系。元素及其关系在内存中适合用什么结构存储，是研究数据的存储结构时要考虑的问题。存储结构非常重要，任何一种数据结构基本操作的设计都取决于其逻辑结构，但基本操作的实现就要依赖于数据的存储结构。

常见的存储结构有顺序存储和链式存储两种。顺序存储是用一块连续的空间来存储数据，同时借助这组空间在地址上的邻接及有序性来存储元素之间的关系，顺序存储结构称顺序结构。高级编程语言中的数组可以实现连续空间的获取，实现顺序结构。如在内存中使用一个数组来存储一个队列，队列中的元素有你先我后的关系，当把队列中元素存储在数组中时，可以让队列元素的先后和数组下标的顺序保持一致，例如队列中的第一个元素存储在数组的 0 下标分量中，队列中第二位元素存储在下标为 1 的数组分量中，依此类推。链式存储是在内存中使用多个独立的内存空间，每个独立的空间除了包括存储元素的空间外，还包括存储元素之间关系的附加的空间。链式存储的数据结构也称链式结构。如队列用链式结构，先声明一个结构数据类型，结构类型的每个变量称作一个结点，结点中含有一个元素字段和一个指针字段，每个结点都存储了队列中的一个元素和指向存储队列中下一个元素结点的指针。

1.1.5 基本操作的实现

当数据在内存中以某种结构存储后，就要研究这种数据结构的基本操作实现方法。基本操作的设计依赖于逻辑结构，而实现要依赖于物理结构。实现方法的设计从存储结构出发，如某种顺序结构，其操作就是对数组的不同目的的操作，具体实现方法因人而异，目标都是要符合逻辑结构中对操作的条件和结果定义。

1.1.6 典型应用

每一种数据结构都是现实问题的抽象，都对应着一些最适合解决的问题。如线性结构中的栈，可以解决高级编程语言程序编译中的符号匹配检查、表达式计算问题；图结构可以解决城市之间的最经济航线问题以及工程项目中的工期和关键活动问题等。

1.2 数据结构的 C 语言实现

数据结构是关于元素及元素间关系的分析、处理，原则上并不固定依赖于任何高级编程语言，所以有些数据结构书籍全部使用伪代码，在实际应用中实现时再换成具体的高级编程语言。本书直接采用 C 语言描述，用 C 语言编程，便于读者直接运行、验证、使用。

C 语言是一种历史悠久且使用范围广泛的语言，也更适应有 C 语言编程基础的相关专业的学生。当然，根据实际工作中开发工具、语言的需求，也可以方便地改为其他语言。

数据结构研究具有一定关系的、类型相同的一组元素，并不特定指向某种具体类型，如整型、字符型或者更加复杂的结构类型。但无论何种类型，它们在关系、基本操作处理方法上都是相同的，因此在分析某种数据结构时，假定它是一种抽象的数据类型 elemType。在算法的 C 语言实现中，可以根据问题中具体的类型，事先利用类型定义将 elemType 类型具体化。如语句“typedef int elemType;”就是将 elemType 类型具体化为整型。

本书使用了 C 语言中最基本的核心部分，包括输入/输出、数据类型、变量、表达式、算术运算、逻辑运算、分支语句、循环语句、函数、递归函数、数组、指针及结构化程序设计等。

1.3 算法及算法分析

1.3.1 算法及其要求

1. 算法

数据结构除了研究元素及元素关系的存储，还要研究在某种存储结构下基本操作的实现。例如，用数组存储的队列如何实现进队、出队操作？设计好的出队的具体方法就是算法。算法就是解决一个具体问题的方法和步骤，它必须满足如下 5 个特性。

算法

（1）有穷性：一个算法中的每一步要在有限的时间内完成，而整个算法必须在有限步之后完成。

（2）确定性：算法中的每一步都有确定的含义，没有二义性。

（3）可行性：算法中的每一步都是经过有限次基本操作就可以完成的，每一步自身没有复杂的算法问题。

（4）有输入：根据问题需要，一个算法可以有零个或者若干个输入作为解决问题的已知条件。

（5）有输出：算法执行结束后，有零个或者若干个输出作为算法运行结果。

2. 算法的要求

即使是同一个问题，解决的方法也会因人而异、千差万别，即算法是不一样的。如何度量并比较不同算法的优劣？通常对一个算法可有以下几个方面的考量。

（1）正确性：准确反映并能满足具体问题的要求，具体来说就是对于任意一个合法的输入，经过算法执行之后应能给出正确的结果。

（2）可读性：指可供人们阅读的容易程度。可读性好的算法能利于人们的阅读、理解、交流，也便于设计者进行调试和纠错。

（3）健壮性：对各种不同的输入都要有相应的反应。如果输入数据合法，就要有相应的输出；如果输入数据不合法，也要有相应的响应处理，如提示信息输出，而不是任由系统发出非法错误警告。

（4）时间效率：算法的执行时间，该时间应能满足问题解决的时间容忍要求，如一些实时系统对处理时间有一个及时性反应的要求。如果有多个算法，执行时间越短，算法的时间效率越高。

（5）空间效率：算法执行期间所需要的最大内存空间。所需要的内存空间越少，则空间效率越高。

3. 常见错误

生活中处理问题的方法并不都能转换为算法，如排队问题，描述的方法是“前看看后看看，比我高的我站他后面，比我矮的站我前面”，这样几分钟后队伍就按照个头高矮排好了。这个排序方法按照算法的定义要求，显然有很多问题，首先就是模糊不精确。要精确地描述出使用的方法似乎很难。按照这个思维去设计算法就会发现，根据生活中积累的处理问题的经验似乎很难直接转换为一个算法。这就是“看别人的算法都懂，自己设计算法就不知道该怎么做了”，因为遇到问题时人们总是下意识地首先想到以往的经验，在此基础上找解决办法，可惜生活中并没有积累这样的经验。

算法中的基本操作应该是计算机语言能够处理的操作，如赋值、算术运算、比较运算、输入/输出等基本运算。例如，排序问题显然主要依赖于比较和交换，而基本运算中的比较运算是一个二元操作，冒泡排序方法就是利用两两比较、两两交换（t=a; a=b; b=t; 3 个赋值完成两个变量交换）最终完成了排序任务。

1.3.2 时间复杂度的度量

算法的时空复杂度

算法的执行时间是指依据算法编制的程序运行时所消耗的时间，度量方法有运行后度量和运行前分析。运行后度量指根据不同算法事先编制好的程序和同样的测试数据，在程序运行时借助机器的计时功能进行计时，当不同程序运行结束时，分别记录实际的运行时间并进行比较。运行前分析指在算法设计好但还没有编程实现时，根据几个方面的影响因素对算法的执行时间进行分析。前者受机器性能、数据分布和运行时的软件环境影响较大，有时会掩盖一些算法的不足，因此一般采用运行前分析法。

分析时间复杂度时所要注意的影响因素如下。

（1）机器的运算速度。这里主要考虑计算机的主频和字长。主频描述了 CPU 内数字脉冲信号震荡的速度，一般来说，主频高，则运算快。字长是运算器能够并行处理的，也是存储器每次读写操作时所能包含的二进制码的位数，如 16 位、32 位、64 位。一般来说，字长越长，处理速度越快，精确度越高。

（2）编译后代码的质量。高级编程语言编制的程序通常要进行编译，不同编译器往往采用不同的优化策略，所以代码运行效率不同，也称代码质量不同。

（3）书写程序所用的语言。同样的算法，使用的编程语言越高级，实现的效率就越高，但执行的效率就越低。

（4）问题的规模。规模即指问题涉及的数据量的大小，通常数据量越大效率越低，一旦数据量大到一定级别，可能就要用到大数据处理方法，如分布式。

（5）算法采用的方法和策略。对于同一问题可以有不同的方法和策略进行解决，算法设计常用的技术有迭代法、枚举法、贪心法、回溯法等，同一算法采用不同的方法和策略消耗的时间可能就不同。

在以上的影响因素中，（1）～（4）都和软硬件环境、问题本身特点有关。设计算法时，一般无从得知将要使用的机器和数据的具体情况，因此这里的时间消耗分析主要是从算法的设计方案入手，即（5）是设计算法时需要特别考虑的。为了量化时间效率，可以用估计算法的运行时间来度量算法策略的优劣。理论上说，运行时间是算法实现中所有语句执行的时间总和，由于不同机型指令集不同，且不同指令执行

时间也不同，所以使用算法中语句执行的次数来度量更合理，语句执行的次数越多，时间花费越长。语句执行次数称时间频度，一组相同的数据可能引起一些语句反复执行，因此一般算法的时间频度和要处理的数据规模有关，故可以表示为数据规模 n 的函数 $T(n)$。以下是一个累加器算法的时间频度计算程序代码块。

```
s = 0;
for (i=0; i<n; i++)
    s = s+ i;
```

分析上述算法，语句 s = 0 执行 1 次，i=0 执行 1 次，i<n 执行 n+1 次，i++执行 n 次，s = s+i 执行 n 次，共计执行 3n+3 次。因此这段算法的时间频度为 $T(n)$=3n+3。

对于各种不同的语句，只要是编程语言中的基本语句，如变量声明、赋值语句、输入语句、输出语句等，都忽略其不同，将其视作一条标准语句，时间频度就是标准语句的执行次数；对于循环语句，需要计算其实际运行的次数，而不是看语句书写的次数；对于分支语句，以执行语句最多的那个分支计算。例如如下代码块。

```
if (n>0)
{   for (i=0; i<n; i++)
        printf("%d ", i);
}
else
    printf("n<=0!");
```

这里语句的执行次数计算 n>0 的情况，为 n 次，而非 n<=0 的 1 次。

当数据规模很大时，算法消耗的时间会是怎样的走势？如果 n 趋于无穷时，$T(n)/f(n)$的极限为一个非零常数，则称 $O(f(n))$为算法的渐进时间复杂度，简称时间复杂度。

即存在 $C \in R$，$N_0 \in N$，当 $n>N_0$ 时，$T(n) \leqslant C*f(n)$。

假设有如下表达式。

$$T(n)=3n^2+2n+10$$

$$T(n) \leqslant 3n^2+2n^2+10n^2 \leqslant 15n^2$$

显然存在 N_0=1，C=15，当 $n > N_0$ 时，有 $T(n) \leqslant 15*n^2$，即 $f(n)=n^2$，算法的复杂度为 $O(n^2)$。

同理，当 $T(n)$=3n+3 时，$T(n) \leqslant 6n$，时间复杂度为 $O(n)$。

可以看出，时间复杂度是一个时间频度表达式中的最高次项，且不带系数。

$T(n)$=3n+3 的时间复杂度和 $T(n)=n$ 的时间复杂度一样，都是 $O(n)$，这说明和数据规模 n 无关的常数项 3 不影响时间复杂度，最高次项 3n 中的系数 3 也不起作用，因此在计算时间复杂度时，如果语句的执行次数和数据规模 n 的变化没有关系，这样的语句可以不计入内，如累加器示例中的语句“s=0; i=0;”。通常数据规模对执行语句的重复次数的影响都在循环语句中，而循环控制条件和循环变量的变化次数和循环体的执行次数接近一致，故只需要看循环体的执行次数即可。

再看如下示例代码块。

```
s=0;
for (i=0; i<n; i++)
    for (j=0; j<i; j++)
        {  s=s+i+j;
           printf("s=%d", s);
        }
```

这个内循环体执行多少次呢？内循环体执行次数受外循环变量 i 值的控制，也就是说间接和问题规模 n 有关。这种情况可以将外循环打开来算：当 i=0 时，内循环体执行 0 次；当 i=1 时，内循环体执行 1 次；当 i=n−1 时，内循环体执行 n−1 次；所以内循环体一共执行 0+1+2+…+n−1=n(n−1)/2 次，时间复杂度为

$O(n^2)$。而内循环体内有两个语句，执行语句次数变为 $n(n-1)$，但这只是改变了时间频度函数的系数，不影响时间复杂度的结果，故循环体内的语句如果不再含循环，具体条数也不用细算。

时间复杂度的计算方法总结如下：找到算法中和数据规模 n 有关的循环语句，计算循环体的执行次数获得时间频度函数；之后观察时间频度函数中关于 n 的最高次项，去掉其系数，即是时间复杂度函数。

特殊地，如果算法中无执行次数和数据规模 n 相关的语句，即时间频度函数是一个常量，则时间复杂度为 $O(1)$。常见算法的时间复杂度有常量阶 $O(1)$、线性阶 $O(n)$、对数阶 $O(\log_2 n)$、线性对数阶 $O(n\log_2 n)$、平方阶 $O(n^2)$、立方阶 $O(n^3)$、幂阶 $O(2^n)$、阶乘阶 $O(n!)$、N 幂阶 $O(n^N)$。按照这个顺序排列，时间效率由高到低。一般来说，到达立方阶之后，一旦数据规模大些，时间上就已经是不能忍受了，算是一个顽性算法了。

1.3.3 空间复杂度的度量

算法的空间消耗包括 3 个方面，一是实现算法的程序本身需要占据存储空间；二是待处理的数据需要在内存中存储，占据一定的空间；三是在处理数据的过程中需要一些额外的辅助空间。通常一和二是不可避免的，在设计算法时主要关注额外的辅助空间。渐进空间复杂度也称空间复杂度，和时间复杂度类似，是当数据规模 n 趋于无穷时的辅助空间量阶，计为 $S(n)=O(f(n))$。

分析如下两个实现数据序列逆置的算法程序示例。

```
int a[10]={1,6,2,5,8,9,5,4,3,12};
int t, i;
for (i=0; i<5; i++)
{     t=a[i];
      a[i]=a[10-i-1];
      a[10-i-1]=t;
}
```

这个例子中，为了完成 n=10 个元素的逆置，将 a[0]和 a[9]交换，再将 a[1]和 a[8]交换，最后 a[4]和 a[5]交换，期间使用的辅助空间为一个变量 t，故其空间复杂度和元素个数没有关系，为 $O(1)$；算法的时间频度为 $n/2$，时间复杂度为 $O(n)$。

```
int a[10]={1,6,2,5,8,9,5,4,3,12},b[10];
int t, i;
for (i=0; i<10; i++)
      b[i] = a[10-i-1];
for (i=0; i<10; i++)
      a[i] = b[i];
```

这个例子中，为了完成 n 个元素的逆置，使用了具有 n 个元素空间的数组 b 作为辅助空间，最后也能完成逆置，但空间复杂度为 $O(n)$。算法的时间频度为 $2n$，时间复杂度为 $O(n)$。

一般来说，在内存足够大的情况下，算法更加注重时间效率，仅当算法的时间复杂度一致的情况下才比较空间复杂度的优劣。

1.4 小结

数据元素及元素间的关系称作数据结构。数据结构研究具有某种制约关系的一组元素及元素间关系在内存中如何存储、在各种存储方式下关系操作如何实现以及各种数据结构的典型应用，具体研究分为逻辑结构、物理结构、基本操作实现及典型应用 4 个方面。在分析逻辑结构时，要完全脱离计算机，而仅仅依赖现实元素特征，分析元素、元素关系及基本操作，给出用伪代码描述的抽象数据类型。在物理

结构分析阶段，讨论元素及元素关系在内存中如何存储，存储可以分顺序存储和链式存储。顺序存储使用一块连续的空间存储元素和元素之间的关系；链式存储使用各自独立的空间存储每个元素，并在每个独立的空间中附加字段以存储元素之间的关系部分。在基本操作实现分析阶段，研究在各种存储结构中基本操作的实现方法和步骤，即算法，对于算法提出了时间复杂度和空间复杂度的概念及计算方法，并以此为依据对不同算法进行性能比较。在典型操作阶段，分析所研究的数据结构最适合的实际应用问题。

1.5 习题

1. 什么是数据结构？数据结构有几种类型？
2. 数据结构的研究对象是什么？
3. 什么是逻辑结构和物理结构？
4. 什么是抽象数据类型？用何种语言描述抽象数据类型？
5. 数据基本操作的实现和逻辑结构、物理结构的关系是什么？
6. 我们在现实社会中解决问题的方法和步骤都能称为算法吗？
7. 继程序设计之后学习数据结构对实际问题的解决有什么帮助？
8. 根据以下时间频度函数分别给出时间复杂度。

   $6x^2+3x+5$，$2n\log_2 n+x^3$，$3+7n^2$，16
9. 设 n 是描述问题规模的非负整数，请分别给出下列程序片段的时间复杂度。

```
(1) sum = 0;
    for (i=0; i<n; i++)
      for (j=0; j<n; j++)
        sum = sum+i*j;
(2) sum = 0;
    for (i=1; i<n; i=2*i)
      for (j=0; j<n; j++)
        sum = sum+i*j;
(3) x=1
    while (x<n/2)
    x=2*x;
```

10. 用递归方法计算求 $n!$，并计算其时间频度函数和时间复杂度。
11. 编程计算 $S=1-2+3-4+5-6+\cdots+N$，$N>0$，要求分别用 $O(n)$、$O(1)$时间复杂度来实现。

Chapter 02

第2章

线性表

■ 线性结构是一种简单、常见的数据结构。日常生活中我们经常遇到的，如体育课上学生排成的一列、食堂某个窗口等待买饭的学生排成的一队、教师讲台上的一摞作业本，都呈现出具有线性关系的结构。

线性结构可被确切定义为：一个含有限数量且具有相同特征的元素构成的集合。该集合或者为空，或者仅有一个被称为首元素的元素；仅有一个被称为尾元素的元素；除了尾元素每个元素有且仅有一个直接后继元素；除了首元素，每个元素有且仅有一个直接前驱元素。

常见的几种线性结构有：线性表（List）、时间有序表(Chronological Ordered List)、排序表(Sorted List)、频率有序表（Frequency Ordered List）等。其中线性表，是仅仅通过元素之间的相对位置来确定它们之间相互关系的线性结构。如元素序列 a_1，a_2，a_3，…，a_i，a_{n-1}，a_n，其中 $1 \leqslant i \leqslant n$，$n$ 为元素的个数，且 $n \geqslant 0$。在这个序列中，当 n=5 时，a_1 是首元素，a_5 是尾元素，a_3 的直接前驱元素是 a_2，a_3 的直接后继元素是 a_4。时间有序表、排序表、频率有序表都可以看作线性表的推广。时间有序表是按照元素到达结构的时间先后，来确定元素之间关系的。如，在红灯前停下的一长串汽车，这些汽车构成了一个队列。其中最先到达的为首元素，最后到达的为尾元素；在离开时最先到达的汽车将最先离开，最后到达的将最后离开。后续我们将介绍的栈和队列都是时间有序表。频率有序表是按照元素的使用频率确定它们之间相互关系的；排序表是根据元素的关键字值来加以确定的。其中线性表、栈和队列是最常用的 3 种线性结构，我们将重点加以讨论。排序表和频率有序表将放入以后各章中进行分析和讨论。

2.1 线性表的定义及 ADT


线性表

线性表是一种仅由元素的相互位置确定它们之间相互关系的线性结构，可以从首元素开始用一个自然数表示的序号来标识每一个元素，这样每个元素的位置和元素的序号是一一对应的。当在线性表中插入一个元素之后，线性表的元素个数将随之增大，在插入位置之后的所有元素的序号也将增大 1。删除操作是类似的。插入和删除可以在线性表的任何有效位置上进行：如插入可以在首元素和尾元素之间的任何位置上进行，包括插入在首元素之前或尾元素之后；删除可以针对首元素和尾元素之间的任何元素，包括首、尾元素。

线性表的规模或长度是指线性表中元素的个数。当元素的个数为零时，该线性表称为空表。ADT 2-1 给出了线性表 List 的抽象数据类型，抽象数据类型包括元素、元素之间的关系、日常生活中的各种基本操作，以及进行各种基本操作必须满足的前提和得到的结果。

ADT 2-1：线性表 List 的 ADT。

```
Data: { xi | xi∈ ElemSet, i=1,2,3,…n, n > 0} 或 Φ; ElemSet为元素集合。
Relation: {<xi,xi+1>|xi,xi+1∈ElemSet, i=1,2,3,…n-1}, x1为首元素, xn为尾元素。
Operations:
    initialize
        前提:    无或指定List的规模。
        结果:    分配相应空间及初始化。
    isEmpty
        前提:    无。
        结果:    表List为空返回1，否则返回0。
    isFull
        前提:    无。
        结果:    表List为满返回1，否则返回0。
    length
        前提:    无。
        结果:    返回表List中的元素个数。
    get
        前提:    表List非空且已知元素序号。
        结果:    返回相应元素的数据值。
```