全国
计算机等级考试
专业辅导用书

本书附赠由虎奔教育提供的学习卡一张

2015年考试专用

根据教育部最新大纲编写

National Computer Rank Examination

全国计算机等级考试

无纸化专用教材

二级公共基础知识

刘欣苗 编著

全国计算机等级考试命题研究室 虎奔教育教研中心 审定

手机版学习软件

题库试题，一网打尽，覆盖99%最新真考题库
按关键字快速查找指定试题，随时随地查看解析，你懂的

虎奔科举网

一学就懂，学完就会的课程
10分钟1个知识点，所学即所考
全职老师在线答疑，不懂就问
量身制定学习计划，定时催促学习进度

赠428元
等考大礼包
手机软件
PC版软件
虎奔科举网体验班和优惠券
随身学

清华大学出版社

全国计算机等级考试专业辅导用书

全国计算机等级考试
无纸化专用教材

二级公共基础知识

刘欣苗　编著

清华大学出版社
北京

内 容 简 介

本书严格依据最新颁布的《全国计算机等级考试大纲》编写，并结合了历年考题的特点、考题的分布和解题的方法。

本书分为4章，包括数据结构与算法、程序设计基础、软件工程基础及数据库设计基础等内容。

本书适合报考全国计算机等级考试二级科目的考生选用，也可作为大中专院校相关专业的教学辅导用书或相关培训课程的教材。

图书在版编目(CIP)数据

全国计算机等级考试无纸化专用教材. 二级公共基础知识 / 刘欣苗编著. —北京：清华大学出版社，2015
全国计算机等级考试专业辅导用书
ISBN 978-7-302-38565-3

Ⅰ.①全… Ⅱ.①刘… Ⅲ.①电子计算机－水平考试－自学参考资料 Ⅳ.①TP3

中国版本图书馆CIP数据核字(2014)第273612号

责任编辑： 袁金敏
封面设计： 傅瑞学
责任校对： 徐俊伟
责任印制： 李红英
出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn，http://www.wqbook.com
地　　址： 北京清华大学学研大厦A座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者： 三河市君旺印务有限公司
装 订 者： 三河市新茂装订有限公司
经　　销： 全国新华书店
开　　本： 185mm×260mm　　**印　　张：** 7.5　　**字　　数：** 203千字
版　　次： 2015年1月第1版　　**印　　次：** 2015年1月第1次印刷
印　　数： 1～5000
定　　价： 19.00元

产品编号：062206-01

前　言

全国计算机等级考试(National Computer Rank Examination,NCRE)是经原国家教育委员会(现教育部)批准,由教育部考试中心主办,面向社会,用于考查应试人员计算机应用知识与技能的全国性计算机水平考试体系。计算机等级考试相应证书的取得,已经逐渐成为考生计算机操作水平的衡量标准,另外,也可以为考生以后的学习和工作打下良好的基础。

随着教育信息化步伐的加快,按教育部要求,从2013年上半年开始,全国计算机等级考试已经完全采用无纸化考试的形式。为了使教师授课和考生备考尽快适应考试形式的变化,本书编写组组织具有多年教学和命题经验的各方专业人士,结合最新考试大纲,深入分析最新无纸化考试形式和题库,精心编写了本套无纸化专用教材。本书具有以下特点。

1. 知识点直击真考

深入分析和研究历年考试真题,结合最新考试大纲和无纸化考试的命题规律,知识点的安排完全依据真考考点,并将典型真考试题作为例题讲解,使考生在初学时就能掌握知识点的考试形式。

2. 课后题查缺补漏

为巩固考生对重要知识点的把握,本书每章均配有课后习题。习题均出自无纸化真考题库,具有典型性和很强的针对性。

3. 无纸化真考环境

本书配套软件完全模拟真实考试环境,其中包括4大功能模块:选择题、操作题日常练习系统,强化练习系统,完全仿真的模拟考试系统以及真人高清名师讲堂系统。同时软件中配有所有试题的答案,方便有需要的考生查阅或打印。

4. 自助式全程服务

虎奔培训、虎奔官网、手机软件、YY讲座、虎奔网校、免费答疑热线、专业QQ群等互动平台,随时为考生答疑解惑;考前一周冲刺专题,还可以通过虎奔软件自动获取考前预测试卷;考后第一时间点评专题,帮助考生提前预测考试成绩。

本书共4章,由刘欣苗编写,全国计算机等级考试命题研究室和虎奔教育教研中心联合审定。在本书的编写和出版过程中,得到了众多一线教师的大力支持,在此表示衷心的感谢。

由于时间仓促,书中难免存在疏漏之处,我们真诚希望得到广大读者的批评指正。

本书由刘欣苗担任主编,并完成了1、2章的主要编写工作和全书的统稿工作,李媛、李鹏担任副主编,李媛完成第3章的编写工作,李鹏完成第4章的编写工作。参与本书编著工作的还有刘爱格、王希更、路谨铭、王小平、张永刚、石永煊、戚海英等。

编　者

目 录

第 1 章 数据结构与算法

1.1 算法 …… 1
1.1.1 什么是算法 …… 1
1.1.2 算法复杂度 …… 4
1.2 数据结构的基本概念 …… 5
1.2.1 什么是数据结构 …… 5
1.2.2 数据结构的图形表示 …… 7
1.2.3 线性结构与非线性结构 …… 7
1.3 线性表及其顺序存储结构 …… 8
1.3.1 线性表的基本概念 …… 8
1.3.2 线性表的顺序存储结构 …… 8
1.3.3 线性表的插入运算 …… 9
1.3.4 线性表的删除运算 …… 10
1.4 栈和队列 …… 11
1.4.1 栈及其基本运算 …… 11
1.4.2 队列及其基本运算 …… 13
1.5 线性链表 …… 15
1.5.1 线性链表的基本概念 …… 15
1.5.2 线性链表的基本运算 …… 18
1.5.3 循环链表及其基本运算 …… 18
1.6 树和二叉树 …… 19
1.6.1 什么是树结构 …… 19
1.6.2 二叉树及其基本性质 …… 21
1.6.3 二叉树的存储结构 …… 23
1.6.4 二叉树的遍历 …… 25
1.7 查找技术 …… 26
1.7.1 顺序查找 …… 26
1.7.2 二分法查找 …… 27
1.8 排序技术 …… 27
1.8.1 交换类排序法 …… 27
1.8.2 插入类排序法 …… 30
1.8.3 选择类排序法 …… 32
1.8.4 排序方法比较 …… 33
本章小结 …… 34
巩固练习 …… 34

第 2 章 程序设计基础

2.1 程序设计方法与风格 …… 36
2.2 结构化程序设计 …… 38
2.2.1 结构化程序设计的原则 …… 38
2.2.2 结构化程序设计的基本结构 …… 38
2.2.3 结构化程序设计原则和方法的应用 …… 40
2.3 面向对象的程序设计 …… 41
2.3.1 面向对象方法的基本概念 …… 41
2.3.2 面向对象方法的优点 …… 44
本章小结 …… 46
巩固练习 …… 46

第 3 章 软件工程基础

3.1 软件工程基本概念 …… 47
3.1.1 软件定义与软件特点 …… 47
3.1.2 软件危机与软件工程 …… 48
3.1.3 软件工程过程与软件生命周期 …… 49
3.1.4 软件工程的目标与原则 …… 50
3.1.5 软件开发工具与软件开发环境 …… 51

3.2 结构化分析方法 …… 52
3.2.1 需求分析与需求分析方法 …… 52
3.2.2 结构化分析方法 …… 53
3.2.3 软件需求规格说明书 …… 56
3.3 结构化设计方法 …… 57
3.3.1 软件设计概述 …… 57
3.3.2 概要设计 …… 60
3.3.3 详细设计 …… 64
3.4 软件测试 …… 67
3.4.1 软件测试的目的 …… 67
3.4.2 软件测试的准则 …… 67
3.4.3 软件测试技术和方法综述 …… 68
3.4.4 软件测试的实施 …… 73
3.5 程序的调试 …… 76
3.5.1 程序调试的基本概念 …… 76
3.5.2 程序调试方法 …… 77
本章小结 …… 79
巩固练习 …… 79
第 4 章 数据库设计基础
4.1 数据库系统的基本概念 …… 81
4.1.1 数据、数据库、数据库管理系统 …… 81
4.1.2 数据库系统的发展 …… 85
4.1.3 数据库系统的基本特点 …… 86
4.1.4 数据库系统体系结构 …… 87
4.2 数据模型 …… 90
4.2.1 数据模型的基本概念 …… 90
4.2.2 E-R 模型 …… 91
4.2.3 层次模型 …… 94
4.2.4 网状模型 …… 95
4.2.5 关系模型 …… 95
4.3 关系代数 …… 98
4.3.1 关系代数的基本操作 …… 98
4.3.2 关系模型的基本运算 …… 98
4.3.3 关系代数的扩充运算 …… 100
4.3.4 关系代数的应用实例 …… 103
4.4 数据库设计与管理 …… 104
4.4.1 数据库设计概述 …… 104
4.4.2 数据库设计的需求分析 …… 105
4.4.3 数据库概念设计 …… 106
4.4.4 数据库逻辑设计 …… 109
4.4.5 数据库物理设计 …… 111
4.4.6 数据库管理 …… 111
本章小结 …… 112
巩固练习 …… 112
附录 巩固练习参考答案 …… 114

第 1 章　数据结构与算法

数据结构与算法是学习程序设计及应用的基础。本章将详细介绍算法、多种类型的数据结构以及查找和排序技术等重要内容。

1.1　算　法

1.1.1　什么是算法

算法是对解题方案准确而完整的描述。算法是一个十分古老的研究课题，人们对于算法的研究已经有数千年的历史。计算机的出现，为这个课题注入了青春和活力，人们可以将算法编写成程序交给计算机执行，使许多原来认为不可能完成的算法变得可行。值得注意的是，算法并不等于程序，也不等于计算机方法，由于在编程时要受到计算机系统运行环境的限制，所以程序的编制不可能优于算法的设计。算法是指令的有限序列，其中每一条指令表示一个或多个操作。

1. 算法的基本特征

一个算法一般应具有以下几个基本特征。

(1) 可行性

可行性是指算法中的操作都可以通过已经实现的基本运算执行有限次来实现。

例如，在进行数值计算时，如果某计算工具具有 7 位有效数字(如程序设计语言中的单精度运算)，则在计算下列 3 个变量

$$A=10^{12},B=1,C=-10^{12}$$

的和时，如果采用不同的运算顺序，就会得到不同的结果，例如：

$$A+B+C=10^{12}+1+(-10^{12})=0$$

$$A+C+B=10^{12}+(-10^{12})+1=1$$

而在数学上，A＋B＋C 与 A＋C＋B 是完全等价的。因此，算法与计算公式是有差别的。在设计一个算法时，必须考虑它的可行性。

(2) 确定性

确定性是指算法中每条指令都有确切的含义,不存在多义性,一般来说,相同的初始状态下,相同的输入只能得到相同的输出。这一性质也反映了算法与数据公式的明显差别。在解决实际问题时,可能会出现这样的情况:针对某种特殊问题,数学公式是正确的,但按此数学公式设计的计算过程可能会使计算机无所适从。这是因为根据数学公式设计的计算过程只考虑了正常使用的情况,而当出现异常情况时,此计算过程就不能适应了。

(3) 有穷性

算法的有穷性是指一个算法必须在有限的时间内完成,即算法必须能在执行有限个步骤之后终止。另外,算法的有穷性还应包括合理的执行时间,如果一个算法需要执行很长时间甚至上千年才能终止,也是没有意义的。

(4) 拥有足够的情报

一个算法是否有效,还取决于为算法所提供的情报是否足够。因此,一个算法的执行结果总是与输入的初始数据有关,它可以有零个或多个输入,但必须有一个或多个输出。

总之,算法是一个动态的概念,是指一组严谨的定义运算顺序或操作步骤的规则,并且每一个规则都是有效的、明确的,此顺序将在有限的次数下终止。

2. 算法的基本要素

算法由两种基本要素构成:一是对数据对象的运算和操作,二是算法的控制结构。

(1) 算法中对数据对象的运算和操作

指令系统是指一个计算机系统能执行的所有指令的集合。一个算法即为按照题目要求从指令系统中选择合适的指令组成的一组指令序列。因此,算法就是计算机能进行的操作所组成的指令序列。

指令系统是软件与硬件分界的一个主要标志,是软件与硬件之间相互沟通的桥梁。指令系统在计算机系统中的地位如图 1.1 所示。

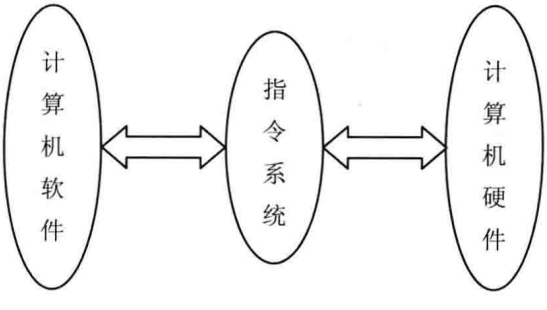


图 1.1 计算机的体系结构

在一般的计算机系统中,基本的运算和操作有以下 4 类:算术运算、逻辑运算、关系运算和数据传输。4 类运算如表 1.1 所示。

表 1.1　4 类基本运算

基本运算	种类
算术运算	加、减、乘、除等
逻辑运算	与、或、非等
关系运算	大于、小于、等于、不等于等
数据传输	赋值、输入、输出等

(2) 算法的控制结构

算法的效果不仅取决于所选用的操作指令，还与各操作之间的执行顺序有关。算法中各操作之间的执行顺序称为算法的控制结构。基本的控制结构包括顺序结构、选择结构和循环结构。

描述算法的工具有传统流程图、N-S 结构化流程图和算法描述语言等。

3. 算法设计的基本方法

算法设计的基本方法有列举法、归纳法、递推法、递归法、减半递推技术和回溯法等。不同的方法间存在着联系，在实际应用中，不同方法通常会交叉使用。

(1) 列举法

列举法的基本思想是根据提出的问题，列举所有可能的情况，并用问题中给定的条件来检验哪些是必需的，哪些是不需要的。因此，列举法常用于解决“是否存在”或“有多少种可能”等类型的问题，例如，我国古代的趣味数学题：“百钱买百鸡”“鸡兔同笼”等求解不定方程的问题，均可采用列举法进行解决。

(2) 归纳法

归纳法的基本思想是，通过列举少量的特殊情况，经过分析，最后归纳出一般的关系。显然，归纳法比列举法更能反映问题的本质，并且可以解决列举量为无限的问题。从本质上讲，归纳就是通过观察一些简单而特殊的情况，最后总结出一般性的结论。

(3) 递推法

递推法是从已知的初始条件出发，逐次推出所要求的各个中间环节和最后结果，其中初始条件或问题本身已经给定，或是通过对问题的分析与简化而确定。递推的本质也是一种归纳，工程上许多递推关系式实际上是通过对实际问题的分析与归纳而得到的，因此，递推关系式通常是归纳的结果。

递推法在数值计算中是极为常见的。例如，裴波那契数列就是采用递推的方法解决问题的。但是，对于数值型的递推算法必须要注意数值计算的稳定性问题。

(4) 递归法

在解决一些复杂问题或问题的规模比较大时，为了降低问题的复杂程度，通常是将问题逐层分解，最后归结为一些最简单的问题。这种将问题逐层分解的过程，并没有对问题进行求解，而只有解决了最后问题分解成的那些最简单的问题后，再沿着原来分解的逆过程逐步进行综合，才能将问题解决，这就是递归的基本思想。

递归分为直接递归和间接递归两种方法。如果一个算法直接调用自己，则称为直接递归调用；如果一个算法 A 调用另一个算法 B，而算法 B 又调用算法 A，则此种递归称为间接递归调用。递归过程是指将一个复杂的问题归纳为若干个较简单的问题，然后将这些较简单的问题再归结为更简单的问题，这个过程可以一直做下去，直到最简单的问题为止。由此可以看出，递归的基础也是归纳。工程实际中的许多问题和数学中的许多函数都是用递归来定义的。递归在可计算性理论和算法设计中占很重要的地位。

(5) 减半递推技术

实际问题的复杂程度往往与问题的规模有着密切的联系。因此,利用分治法解决这类实际问题是有效的。所谓分治法,就是对问题分而治之。工程上常用的分治法是减半递推技术。

所谓"减半",即将问题的规模减半,而问题的性质不变;所谓"递推",是指重复"减半"的过程。

(6) 回溯法

递推和递归算法本质上是对实际问题进行归纳的结果,而减半递推技术也是归纳法的一个分支。在工程上,有些实际的问题很难归纳出一组简单的递推公式或直观的求解步骤,也不能使用无限的列举。对于这类问题,只能采用"试探"的方法,通过对问题的分析,找出解决问题的线索,然后沿着这个线索进行试探,如果试探成功,就得到问题的解,如果不成功,再逐步回退,换别的路线进行试探。这种方法即称为回溯法。

回溯法在处理复杂数据结构方面有着广泛的应用,如人工智能中的机器人下棋等。

1.1.2 算法复杂度

算法复杂度主要包括时间复杂度和空间复杂度。

(1) 算法的时间复杂度

算法的时间复杂度是指执行算法所需要的计算工作量。

为了能够比较客观地反映出一个算法的效率,衡量一个算法的工作量前提是,不仅要求其与所使用的计算机、程序设计语言以及程序编制者无关,而且还应与算法实现过程中的许多细节无关。为此,可以用算法在执行过程中所需基本运算的执行次数来度量算法的工作量,算法的计算工作量用算法所执行的基本运算次数来度量,而算法所执行的基本运算次数是问题规模的函数,即

算法的工作量 = f(n)(其中 n 是问题规模)

所谓问题的规模就是问题计算量的大小。如 1+2,这是规模比较小的问题,但 1+2+3+...+n(其中 n=10 000),这就是规模比较大的问题。

例如,两个 n 阶矩阵相乘所需要的基本运算(即两个实数的乘法)次数为 n^3,即计算量为 n^3,也就是时间复杂度为 n^3。

例如,在下列 3 个程序段中:

```
① {sum = 0;
   a += b;}
② for(i = 1;i<= n;i ++ )
   {a += b;
   sum += a;}/ * 一个简单的 for 循环,循环体内操作执行了 n 次 * /
③ for(i = 1;i<= n;i ++ )
   for(j = 1;j<= n;j ++ )
   {a += b;
   sum += a;}/ * 嵌套的双层 for 循环,循环体内操作执行了 n² 次 * /
```

在①中,基本运算"a+=b;"只执行一次,重复执行次数为 1。

在②中,由于有一个循环,所以基本运算"a+=b;"执行了 n 次。

在③中,有一个嵌套的双层循环,所以基本运算"a+=b;"执行了 n^2 次。

则这 3 个程序段的时间复杂度分别为 O(1)、O(n)和 $O(n^2)$。

在具体分析一个算法的工作量时,在同一个问题规模下,算法所执行的基本运算次数还可

能与特定的输入有关。即输入不同时，算法所执行的基本运算次数不同。在这种情况下，可以用以下两种方法来分析算法的工作量。

① 平均性态；

② 最坏情况复杂性。

另外，在同一问题规模下，若算法执行所需的基本运算次数取决于某一特定输入数据时，可以用平均性态分析和最坏情况分析两种方法来分析算法的工作量。顾名思义，平均性态分析即输入所有可能的平均值，相应的时间复杂度为算法的平均时间复杂度；最坏情况分析则是以最坏的情况估算算法执行时间的一个上界。一般情况下，后者更为常用。

(2) 算法的空间复杂度

算法的空间复杂度是指执行此算法所需要占用的内存空间，包括 3 部分：算法程序所占的空间、输入的初始数据所占的存储空间以及算法执行过程中所需要的额外空间。其中额外空间包括算法程序执行过程中的工作单元以及某种数据结构所需要的附加存储空间(例如，在链式结构中，除了需要存储数据本身之外，还需要存储链接信息)。如果额外空间量相对于问题规模来说是常数，则称该算法是原地工作的。

实际应用中，为了减少算法所占用的存储空间，通常采用压缩存储技术，以减少不必要的额外空间。

1.2　数据结构的基本概念

1.2.1　什么是数据结构

数据结构是指反映数据元素之间关系的数据元素的集合。而数据元素具有广泛含义，一般来说，现实世界中客观存在的一切个体都可以是数据元素，它可以是一个数字或一个字符，也可以是一个具体的事物，或者其他更复杂的信息。

例如：

- 日常生活中一年四季的名称——春季、夏季、秋季、冬季，可以作为一年四季的数据元素；
- 在家庭成员中表示成员间关系的名称——父亲、儿子、女儿，可以作为表示成员的数据元素。

数据结构作为计算机的一门学科，主要研究和讨论以下 3 个方面的问题：

- 数据集合中各数据元素之间所固有的逻辑关系，即数据的逻辑结构；
- 在对数据进行处理时，各数据元素在计算机中的存储关系，即数据的存储结构；
- 对各种数据结构进行的运算。

在实际应用中，被处理的数据元素一般有很多，而且，作为某种处理，其中的数据元素一般具有某种共同特征。一般情况下，在具有相同特征的数据元素集合中，各个数据元素之间存在某种关系(即联系)，这种关系反映了该集合中数据元素所固有的一种数据结构。在数据处理领域中，通常把数据元素之间这种固有的关系简单地用前后件关系(或直接前驱与直接后继关系)来描述。

例如，在考虑一年四季的时间顺序关系时，“春季”是“夏季”的前件(或直接前驱)，而“夏

季”是“春季”的后件(或直接后继);同样,“夏季”是“秋季”的前件,“秋季”是“夏季”的后件;“秋季”是“冬季”的前件,“冬季”是“秋季”的后件。

又如,在考虑家庭中的长幼级关系时,“父亲”是“儿子”和“女儿”的前件,“儿子”和“女儿”是“父亲”的后件。

前后件关系是数据元素之间最基本的关系。

综上所述,数据结构是指相互有关联的数据元素的集合,它包括数据的逻辑结构和数据的物理结构。换句话说,如果各个数据元素之间是有关联的,我们就认为,这个数据元素的集合是有“结构”的。

(1) 数据的逻辑结构

所谓数据的逻辑结构,是指反映数据元素之间逻辑关系(即前后件关系)的数据结构。它包括数据元素的集合和数据元素之间的前后件关系两个要素,其中,用 D 表示数据元素的集合,用 R 来表示数据元素之间的前后件的关系。即一个数据结构可以表示为 B=(D,R),其中,B 表示数据结构。这就是一个二元关系的表示方式。一般用二元组来反映 D 中各数据元素之间的前后件关系。例如,假设 a 与 b 是 D 中的两个数据,则二元组(a,b)表示 a 是 b 的前件,b 是 a 的后件。这样,在 D 中每两个元素之间的关系都可以用这种二元组来表示。

例如,一年四季的数据结构可以表示成

```
B={D,R}
D={春,夏,秋,冬}
R={(春,夏),(夏,秋),(秋,冬)}
```

又如,家庭成员数据结构可以表示成

```
B=(D,R)
D={父亲,儿子,女儿}
R={(父亲,儿子),(父亲,女儿)}
```

再如,n 维向量 $X=(x_1,x_2,\ldots,x_n)$ 也是一种数据结构,即 $X=\{D,R\}$。其中数据元素集合为 $D=\{x_1,x_2,\ldots,x_n\}$,关系为 $R=\{(x_1,x_2),(x_2,x_3),\ldots,(x_{n-1},x_n)\}$。

根据数据元素之间关系的不同特性,通常有 4 类基本的逻辑结构:集合结构、线性结构、树形结构和图形结构。

① 集合结构。在集合结构中,数据元素之间的关系是“属于同一个集合”,集合是元素关系极为松散的一种结构,如图 1.2(a)所示。

② 线性结构。数据元素之间存在着一对一的关系,如图 1.2(b)所示。

③ 树形结构。数据元素之间存在着一对多的关系,如图 1.2(c)所示。

④ 图形结构。数据元素之间存在着多对多的关系,图形结构也称作网状结构,如图 1.2(d)所示。

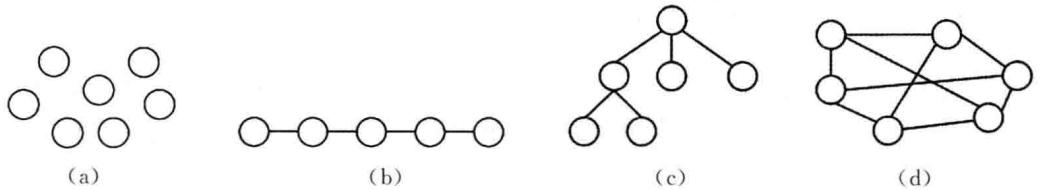


图 1.2 4 类基本结构

(2) 数据的存储结构

数据的逻辑结构可以看成是从具体问题抽象出来的数学模型，它与数据的存储无关。我们研究数据结构的目的是在计算机中实现对它的操作，为此还需要研究如何在计算机中表示一个数据结构。数据的存储结构又称为数据的物理结构，是指数据的逻辑结构在计算机存储空间中的存放方式。

因为数据元素在计算机中存放的位置很可能与逻辑关系不一样，所以在存储数据时，不仅要存放数据的信息，还要存放数据间前后件的信息，只有这样，才能更好地表示计算机存储空间中数据之间的逻辑关系。

数据结构的存储方式包括顺序存储、链式存储、索引存储和散列存储等。而采用不同的存储结构，其数据处理的效率是不同的，所以在进行数据处理时，选择合适的存储结构就显得十分重要。

1.2.2　数据结构的图形表示

一个数据结构除了用二无关系表示外，还可以用图形来表示。用中间标有元素值的方框来表示数据元素，此方框一般称为数据结点，简称结点。对于每一个二元组，需用一条有向线段从前件指向后件。

例如，一年四季的数据结构可以用如图 1.3 所示的图形来表示。

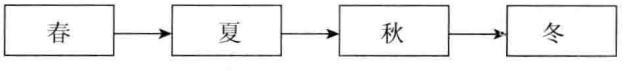


图 1.3　一年四季的数据结构的图形表示

又如，数据结构的知识体系可以用如图 1.4 所示的图形来表示。

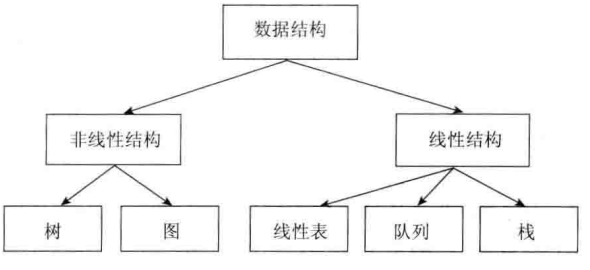


图 1.4　数据结构的知识体系的图形表示

用图形表示数据结构直观易懂，在没有歧义的情况下，前件结点到后件结点连线上的箭头也可以省去。

1.2.3　线性结构与非线性结构

如果在一个数据结构中一个数据元素都没有，则称该数据结构为空的数据结构。在一个空的数据结构中插入一个新的元素后就变为非空；在只有一个数据元素的数据结构中，将该元素删除后就变为空的数据结构。

根据数据结构中各数据元素之间前后关系的复杂程度，一般可将数据结构分为两大类型：线性结构和非线性结构。

若一非空的数据结构有且只有一个根结点，并且每个结点最多有一个直接前驱和一个直

接后继，则称该数据结构为线性结构，也称线性表。

不满足上述条件的数据结构则称为非线性结构。

由此可知，图 1.3 为线性结构，图 1.4 为非线性结构。

线性结构与非线性结构都可以是空的数据结构。同样，空的数据结构既可能是线性结构，也可能是非线性结构，这要根据对该数据结构的运行规则来定，即如果对该数据结构的运算是按线性结构的规则来处理的，则属于线性结构；否则属于非线性结构。

1.3 线性表及其顺序存储结构

1.3.1 线性表的基本概念

在数据结构中，线性表是最简单也是最常用的一种数据结构。

线性表由一组数据元素构成，是由 n 个($n \geqslant 0$) 数据结构元素 $a_1, a_2, a_3, \ldots, a_n$ 组成的一个有限序列。除了表中的第一个元素外，其他所有元素有且只有一个直接前驱；除了最后一个元素外，其他所有元素有且只有一个直接后继。将线性表表示为($a_1, a_2, a_3, \ldots, a_n$)，其中 a_i($i=1,2,\ldots,n$)是线性表的数据元素，也称为线性表的一个结点。线性表中的数据元素必须具有相同的属性，即属于同种数据对象。例如：

字母表(A,B,C,...,Z)就是一个长度为 26 的线性表，其中每个字母就是一个数据元素，如图 1.5 所示。

A	B	C	...	Z

图 1.5 字母表的线性表表示

四季(春，夏，秋，冬)是一个长度为 4 的线性表，其中的每一个季节是一个数据元素。

线性表的基本特征如下：

① 有且仅有一个根结点，无前件；

② 有且仅有一个终端结点，无后件；

③ 除终端结点外，其他所有结点均有且仅有一个后件；

④ 除根结点外，其他所有结点均有且仅有一个前件。

矩阵可以看作一个比较复杂的线性表，在矩阵中，既可以把每一行看成一个数据元素，也可以把每一列看成一个数据元素。其中每一个数据元素(一个行向量或者列向量)实际上又是一个简单的线性表。

1.3.2 线性表的顺序存储结构

在计算机中存放线性表，其最简单的方法是顺序存储，也称为顺序分配。

线性表的顺序储存是指将线性表中的元素在计算机中一段相邻的存储区域中连续存储。

线性表的顺序存储结构具有以下两个基本特点：

① 所有元素所占的存储空间必须是连续的；

② 所有元素在存储空间的位置是按逻辑顺序存放的。

由此可以看出，线性表中元素之间逻辑上的相邻关系即为元素在计算机内物理位置上的相邻关系。所以只要确定了首地址，线性表内所有元素的地址都可以方便地查找出来。

假设线性表中第一个数据元素的存储地址(指第一个字节的地址,即首地址)为 ADR(a_1),每个元素需要占用 k 个存储单元,则线性表中第 i 个元素 a_i 在计算机存储空间中的存储地址为

$$ADR(a_i)=ADR(a_1)+(i-1)\times k$$

即在顺序存储结构中,线性表中每一个数据元素在计算机存储空间中的存储地址由该元素在线性表中的位置序号唯一确定。一般来说,对于长度为 n 的线性表(a_1, a_2,..., a_n),设每个数据元素在内存中占 4 个字节,起始元素的存储地址为 1010,则该线性表在计算机中的顺序存储结构如图 1.6 所示。

存储地址	数据元素
1010	a_1
1013	a_2
…	…
1010+(i-1) ×4	a_i
…	…
1010+(n-1) ×4	a_n

图 1.6　线性表的顺序存储结构示意图

在程序设计语言中,通常定义一个一维数组来表示线性表的顺序存储空间。在用一维数组存放线性表时,该一维数组的长度通常要定义得比线性表的实际长度大一些,以便对线性表进行各种运算,特别是插入运算。在一般情况下,如果线性表的长度在处理过程中是动态变化的,则在开辟线性表的存储空间时要考虑到线性表在动态变化过程中可能达到的最大长度。

对线性表的处理主要有插入、删除、查找、排序、分解、合并、复制、逆转等。线性表的基本运算是插入运算和删除运算,下面两小节主要讨论这两种运算。

1.3.3　线性表的插入运算

线性表的插入运算是指在表的第 i($1\leqslant i\leqslant n+1$)个位置上插入一个新结点 x,使长度为 n 的线性表:

$$(a_1, \ldots, a_{i-1}, a_i, \ldots, a_n)$$

变成长度为 n+1 的线性表:$(a_1, \ldots, a_{i-1}, x, a_i, \ldots, a_n, a_{n+1})$。

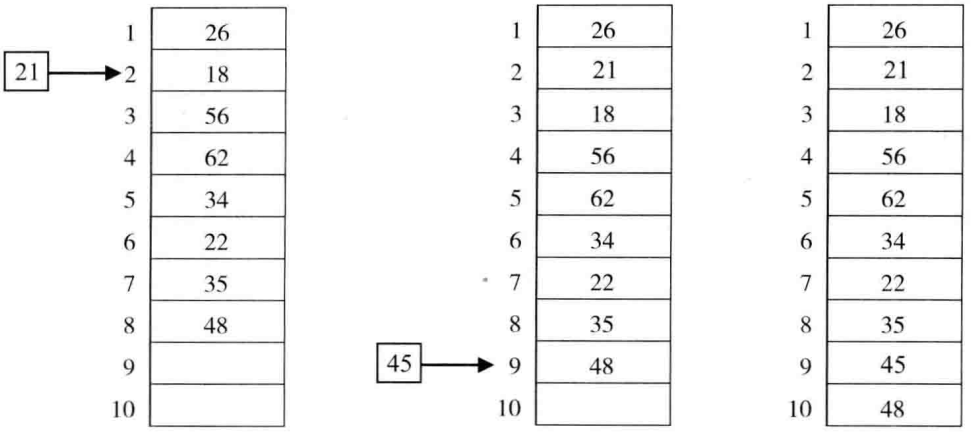
对于一个线性表来说,若在第 i($1\leqslant i\leqslant n+1$)个元素之前插入一个新元素,则若想完成插入操作需以下 3 个步骤:原来第 n 个结点至第 i 个结点依次往后移一个位置;把新结点放在第 i 个位置上;线性表的结点数加 1。例如,图 1.7(a)所示为一个长度为 8 的线性表,其顺序存储在长度为 10 的存储空间中。现在要求在第 2 个元素(即 18)之前插入一个新元素 21。其插入过程如下。

首先从最后一个元素开始直到第 2 个元素,将其中的每一个元素均依次往后移动一个位置,然后将新元素 21 插入到第 2 个位置。

插入一个新元素后,线性表的长度变成了 9,如图 1.7(b)所示。

如果还要在线性表的第 9 个元素之前插入一个新元素 45,则采用类似的方法:将第 9 个元素往后移动一个位置,然后将新元素插入到第 9 个位置。插入后,线性表的长度变成了 10,

如图 1.7(c)所示。



(a)长度为8的线性表　　(b)插入元素21后的线性表　　(c)插入元素45后的线性表

图 1.7　线性表顺序存储结构插入前后的状况

当在线性表尾进行插入运算时,只需在表的末尾增加一个元素,不需要移动线性表中的元素。当在线性表头位置插入新的元素时,则需要移动表中所有的元素。

一般线性表会事先开辟出一个大于线性表长度的空间,但当多次插入运算后,可能会出现在存储空间已满的情况下仍继续进行插入运算的情况,此时将会产生错误,此类错误称为“上溢”。

下面分析插入算法的时间复杂度。顺序表的插入运算,其时间主要花费在数据的移动上,在第 i 个位置插入 b,从 a_i 到 a_n 都要向后移动一个位置,共需要移动 n−i+1 个元素,而 i 的取值范围为 $1 \leqslant i \leqslant n$,即有 n+1 个位置可以插入。假设在第 i 个位置上做插入操作的概率为 P_i,则平均移动数据元素的次数为

$$E_{in} = \sum_{i=1}^{n+1} P_i(n-i+1)$$

设 $P_i = 1/(n+1)$,即为等概率情况,则

$$E_{in} = \sum_{i=1}^{n+1} P_i(n-i+1) = \frac{1}{n+1}\sum_{i=1}^{n+1}(n-i+1) = \frac{n}{2}$$

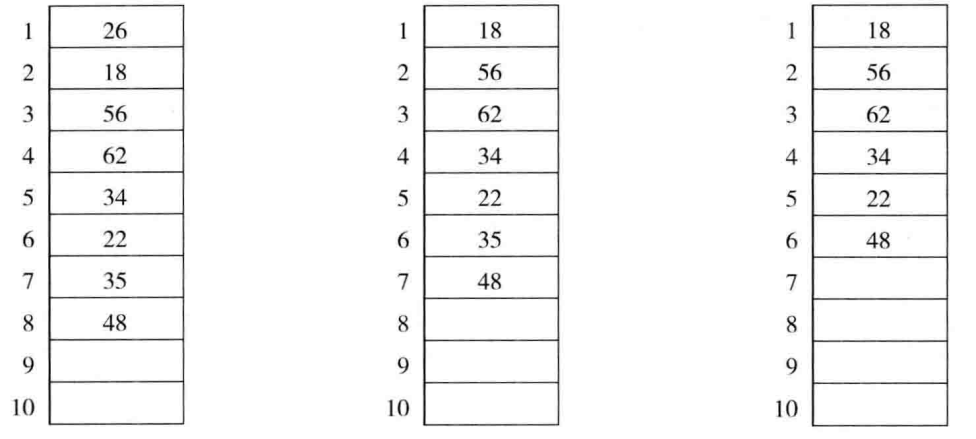
这说明:在顺序表上进行插入操作大约需要移动表中一半的数据元素,显然该算法的时间复杂度为 O(n)。

1.3.4　线性表的删除运算

在对线性表进行删除操作时,若要删除第 i 个元素,需要进行以下步骤:把第 i 元素之后(不包括第 i 个元素)的 n−i 个元素依次前移一个位置;线性表的结点数减 1。例如,图 1.8 (a)所示为一个长度为 8 的线性表顺序存储在长度为 10 的存储空间中,现在要求删除线性表中的第 1 个元素(即删除元素 26)。其删除过程如下:

从第 2 个元素开始直到最后一个元素,将其中的每一个元素均依次往前移动一个位置。此时,线性表的长度变成了 7,如图 1.8(b)所示。

如果还要删除线性表的第 6 个元素，则采用类似的方法：将第 7 个元素往前移动一个位置。此时，线性表的长度变成了 6，如图 1.8(c)所示。



（a）长度为8的线性表　　（b）删除元素26后的线性表　　（c）删除元素35后的线性表

图 1.8　线性表顺序存储结构删除前后的状况

当删除运算在线性表表尾进行时，即删除第 n 个元素，则不需要移动线性表中的元素。当要删除第 1 个元素时，则需要移动表中除第 1 个元素外所有的元素。

下面分析删除算法的时间复杂度。与插入运算相同，其时间主要花费在数据的移动上，当删除第 i 个位置上的元素时，从 a_{i+1} 到 a_n 都要向前移动一个位置，共需要移动 n－i 个元素，而 i 的取值范围为 $1 \leqslant i \leqslant n$，即有 n 个位置可以删除。假设在第 i 个位置上做删除操作的概率为 P_i，则平均移动数据元素的次数为

$$E_{de} = \sum_{i=1}^{n} P_i(n-i)$$

在等概率情况下，$P_i = 1/n$，则

$$E_{de} = \sum_{i=1}^{n} P_i(n-i) = \frac{1}{n}\sum_{i=1}^{n}(n-i) = \frac{n-1}{2}$$

这说明在顺序表上进行删除操作时，大约需要移动表中约一半的元素，显然该算法的时间复杂度也是 O(n)。

1.4　栈和队列

栈和队列是两种特殊的线性结构。从数据结构的角度看，它们是线性表；从操作的角度看，它们是操作受限的线性表。在日常生活中经常会遇到栈或队列的实例，并且栈和队列还广泛应用于各种软件系统之中。

1.4.1　栈及其基本运算

(1) 栈的基本概念

栈是一种特殊的线性表，如图 1.9 所示，其特殊性体现在，插入和删除运算都只能在线性