国家计算机等级考试

二级基础知识教程

主编 姜春茂 黄春梅 范雪琴

哈尔滨地图出版社

国家计算机等级考试二级基础知识教程

GUOJIA JISUANJI DENGJI KAOSHI REJI JICHU ZHISHI JIAOCHENG

主　编　姜春茂　黄春梅　范雪琴

哈尔滨地图出版社
·哈尔滨·

图书在版编目(CIP)数据

国家计算机等级考试二级基础知识教程/姜春茂，黄春梅，范雪琴主编. —哈尔滨：哈尔滨地图出版社，2007.1
ISBN 978-7-80717-560-5

Ⅰ.国… Ⅱ.①姜…②黄…③范… Ⅲ.电子计算机—水平考试—教材 Ⅳ.TP3

中国版本图书馆 CIP 数据核字(2007)第 042779 号

哈尔滨地图出版社出版发行
(地址：哈尔滨市南岗区测绘路 2 号 邮编：150086)
哈尔滨庆大印刷厂印刷
开本：787 mm × 1 092 mm 1/16 印张：8.125 字数：200 千字
ISBN 978-7-80717-560-5
2007 年 1 月第 1 版 2007 年 1 月第 1 次印刷
印数：1 ~ 500 定价：18.00 元

前　言

全国计算机等级考试（National Computer Rank Examination 简称为 NCRE），是经过教育部批准，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能力的全国计算机考试水平体系，从 1994 年开考至今已有 20 多次了。

为了适应新形势下我国市场经济的发展需要，进一步满足人们学习计算机的需求，全国计算机等级考试在科目设置、考核内容、考试形式上进行了调整，启用新大纲，2005 年在全国推广。

调整后增加了二级基础知识，增加二级 JAVA，ACCESS，C++科目，上机环境在 window2000 下进行。

为了适应考试需要，帮助考生做好复习工作，根据教育部考试中心复习大纲，我们编写了一套计算机等级考试辅教材，本套教材特点是：

从大纲出发，紧扣考试题目、题型，重点、难点突出。通过典型例题提高学生学习能力，加强应试能力，取得高分。

通过本套书的编写，我们也感觉到了同学们的迫切心情，使得我们在编书过程中有了很多动力。

本书由姜春茂、黄春梅、范雪琴共同担任主编，参加编写的还有王珊、段莹等同志。

作者

2007 年 1 月

一书在手　考试无忧

祝愿考生学习进步　轻松过关

目　录

第 1 章　基本数据结构与算法

1．1　数据结构概述

随着计算机技术的发展，其应用范围在不断扩大。计算机所处理的数据量在不断扩大，所处理的数据类型已不再是单纯的数值数据，而更多的是非数值数据，如表格、声音、图形和图像等具有一定结构的数据。也就是说，这些非数值数据并不是杂乱无章的，它们一定有内在的联系，只有弄清楚它们之间的本质的联系，才能够对它们的处理显得格外游刃有余，目前数据的逻辑结构、存储结构及对数据的基本操作，算法设计和算法分析等都是非常重要的研究内容。对于软件设计技术的理论基础，“数据结构”就不仅仅是计算机学科的核心课程，也应该是所有应用计算机的其他学科所必须掌握的课程。

1．1．1　数据结构的定义

数据（data）是信息的载体。它能够被计算机识别、存储和加工处理，是计算机程序加工的“原料”。随着计算机应用领域的扩大，数据的范畴包括：数值、字符串、图像和声音等各种多媒体信息。

数据元素（data element）是数据的基本单位。数据元素也称元素、结点、记录。如一年四季的季节命名可以作为季节的数据元素，表示家庭成员的各成员名（父亲、母亲、儿子、女儿），可以作为家庭成员的数据元素。

数据项：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。数据项是具有独立含义的最小标识单位。

数据结构（data structure）指的是数据之间的相互关系，即数据的组织形式。数据结构一般包括以下三方面内容：

1．数据的逻辑结构（logical structure）

数据的逻辑结构是指数据元素之间的逻辑关系。

数据的逻辑结构是从逻辑关系上描述数据，与数据的存储无关，是独立于计算机的。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．数据的存储结构（storage structure）

数据的存储结构是指数据元素及其关系在计算机内存储器内的表示形式。

数据的存储结构是逻辑结构用计算机语言的实现（亦称为映象），它依赖于计算机语言。对机器语言而言，存储结构是具体的。一般只在高级语言的层次上讨论存储结构。

3．数据的操作

数据的运算即对数据施加的操作。

数据的运算定义在数据的逻辑结构上，每种逻辑结构都有一个运算的集合。最常用的检索、插入、删除、更新、排序等运算实际上只是在抽象的数据上所施加的一系列抽象的操作。所谓抽象的操作，是指我们只知道这些操作是“做什么”，而不考虑“如何做”。只有确定了存储结构之后，才考虑如何具体实现这些运算。

1．1．2　数据结构的图形表示

一个数据结构可以用二元关系来表示（D，R），其中 D 表示数据集合中的每一个数据元素，R 表示这些数据元素之间的关系。另外，一个数据结构也可直观地用图形来表示，在数据的图形表示中，对于数据集合 D 中的每一个元素用中间标有元素值的方框（或圆框）来表示，一般称为数据结点（或简称结点），对于关系 R 中的每一个二元组，用一条有向线段从直接前驱指向直接后继。例如，一年四季的数据结点可以用图 1－1 来表示。

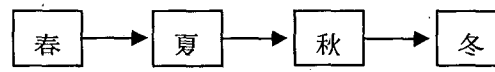

图 1－1　一年四季数据结构的图形表示

1．1．3　线性结构与非线性结构

在不产生混淆的前提下，常将数据的逻辑结构简称为数据结构。数据的逻辑结构有两大类：

1．线性结构

线性结构的逻辑特征是：若线性表是非空集，则有且只有一个开始结点和一个终端结点，并且所有结点都最多只有一个直接前驱和一个直接后继。线性表是一个典型的线性结构。栈、队列、串等都是线性结构。

2．非线性结构

非线性结构的逻辑特征是：一个结点可能有多个直接前驱和直接后继。树和图等数据结构都是非线性结构。

数据的存储结构可用以下四种基本存储方法得到：

（1）顺序存储方法

该方法把逻辑上相邻的结点存储在物理位置上相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来实现。由此得到的存储表示称为顺序存储结构（sequential storage structure），通常借助程序语言的数组描述。

该方法主要应用于线性的数据结构。非线性的数据结构也可通过某种线性化的方法实现顺序存储。

（2）链接存储方法

该方法不要求逻辑上相邻的结点在物理位置上亦相邻，结点间的逻辑关系由附加的指针字段表示。由此得到的存储表示称为链式存储结构（linked storage structure），通常借助于程序语言的指针类型描述。

（3）索引存储方法

该方法通常在存储结点信息的同时，还建立附加的索引表。

索引表由若干索引组成。若每个站点在索引表中都有一个索引项，则该索引表称之为稠密索引（dense index）。若一组结点在索引表中对应一个索引项，则该索引表称为稀疏索引（spare index）。索引项的一般形式为：

（关键字、地址）

关键字是能惟一标识一个结点的那些数据项。稠密索引中索引项的地址指示结点所在的存储位置，稀疏索引中索引项的地址只是一组结点的起始存储位置。

（4）散列存储方法

该方法的基本思想是:根据结点的关键字直接计算出该结点的存储地址。

1．2　算法

数据的运算通过算法（algorithm）来描述 ，因此讨论算法是数据结构课程的重要内容之一。

1．2．1　算法的基本概念

1．算法的定义

当我们要编写一个程序的时候，我们总要首先想好这个程序是干什么的？应该如何实现这些目标？应该先进行什么处理，后进行什么处理？所处理的数据的格式是什么？遇到一些复杂的问题，我们可能还需要考虑采用什么数学方法来处理。这一切都涉及一个专业名词——“算法”。

算法是解题方案准确而完整的描述，它以一个或多个值作为输入，并产生一个或多个值作为输出。因此，我们可以从下面两个方面来理解算法：

（1）一个算法可以被认为是用来解决一个计算问题的工具。

（2）一个算法是一系列将输入转换为输出的计算步骤。

若一个算法对于每个输入实例均能终止并给出正确的结果，则称该算法是正确的。正确的算法解决了给定的计算问题。

一个不正确的算法是指对某些输入实例不终止，或者虽然终止但给出的结果不是所期望得到的答案，一般只考虑正确的算法。

很多时候，程序设计者所面临的问题就是寻找一个合适的算法。例如，一个熟练的程

序员要设计一个下“五子棋”的游戏程序，对他而言，C 语言的编程规则已经清楚。他所面对的核心问题是寻找一种可以模拟人下棋的算法。因此，算法在软件设计中具有重要的地位。正如著名的计算机科学家沃思（Nikiklaus Wirth）所指出的如下公式：

程序＝数据结构＋算法

2．算法的特性

每一个算法具有下列几个特性：

（1）有穷性（finiteness）

一个算法要在有限的步骤内解决问题（这里所说的步骤是指计算机执行步骤）。计算机程序不能无限地运行下去（甚至不能长时间地运行下去），所以一个无限执行的方法不能成为程序设计中的“算法”。

例如，求某一自然数 N 的阶乘：

$N!=1\times2\times3\times\ldots\times N$

这是一个算法。因为对任何一个自然数而言，无论这个数多大，总是有限的。用这个公式计算 $N!$总是需要有限的步骤。

但是，以下计算公式则不能作为算法，因为其计算步骤是无限的：

$SUM=1+1/1+1/2+1/3+\ldots+1/n+\ldots$

事实上有穷性是指合理的范围之内，比如设计了一个算法是有限的，但按照目前计算机发展的水平要计算 1 000 年才能完成，这样的算法没有实际意义，可以不当作算法，可以视为无穷。

实际上，在计算机的需要加密算法中，可以解密的方法不是不存在，而是要执行这样的解密算法需要大量的时间。这样就实现了保密。所谓保密就是在一定的时间内信息不被他人知晓。

当然，计算机技术的进步会对算法产生影响。对于现在的计算机 1 000 年才能完成的算法将来可能在几个月内就能完成，到那时某些现在无穷性的算法将变成切实可行的算法。

（2）确定性（definiteness）

算法中的每一条指令必须有确切的含义，不会产生二义性。

（3）可行性（effectiveness）

一个算法是能执行的，也就是说，算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现。

（4）拥有足够多的情报

一个算法是否有效，还取决于为算法所提供的情报是否够多。一般来说，算法总是要施加到某些对象上的一些运算，总是和某些输入相关，不同的输入将会有不同的输出。当

输入有误时，算法本身也就无法执行或者导致执行有错。

3．算法的描述

一个算法可以用自然语言、计算机程序语言或其他语言来说明，惟一的要求是该说明必须精确地描述计算过程。一般而言，描述算法最合适的语言是介于自然语言和程序语言之间的伪语言。它的控制结构往往类似于 Pascal、C、Java 等程序语言，但其中可使用任何表达能力强的方法使算法表达更加清晰和简洁，而不至于陷入具体的程序语言的某些细节。

4．算法的设计要求

在进行算法的设计时，应满足下列的要求：

（1）正确性

一个算法应当能够解决具体问题。其"正确性"可分为以下几个方面：

①不含逻辑错误；

②几组输入数据能够得出满足要求的结果；

③精心选择的典型、苛刻的输入数据都能得到要求的结果；

④一切合法的输入都能输出满足要求的结果。

（2）可读性

算法应该可以用能够被人理解的形式表示。太复杂的、不能被程序员所理解的算法难以在程序设计中采用。

（3）健壮性

健壮性指算法具有抵御"恶劣"输入信息的能力 。当输入数据非法时，算法也能适当地作出反应或进行处理，而不会产生莫明其妙的输出结果。例如，当输入三个边的长度值来计算三角形的面积时，一个有效的算法是三个输入数据不能构成一个三角形时，应报告输入的错误，同时返回一个表示错误或错误性质的值并中止执行。

（4）效率与低存储量的需求

高效率和低存储量是程序员追求的目标。效率指的是算法执行时间，对于一个问题如果有多个算法可以解决，执行时间短的算法效率高。存储量需求指算法执行进程所需要的最大存储空间。效率与低存储量需求这两者都与问题规模有关。占用存储量最小、运算时间最小的算法就是最好的算法。但是在实际中，运算时间和存储空间往往是一对矛盾，要根据具体情况选择更优先考虑哪一个因素。

1．2．2　算法复杂度

1．算法的时间复杂度

算法求解问题的输入量称为问题的规模（Size），一般用一个整数表示。一个算法的时间复杂度（time complexity，也称时间复杂性）$T(n)$是该算法的时间耗费，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，时间复杂度 $T(n)$ 的数量级（阶）称为算法的渐进时间复杂度，或简称为时间复杂度。最坏情况下的时间复杂度称为最坏时间复杂度。一般不特别说明，讨论的时间复杂度均是最坏情况下的时间复杂度。平均时间复杂度是指所有可能的输入实例均以等概率出现的情况下，算法的期望运算时间。

常见的时间复杂度按数量级递增排列依次为：常数 $O(1)$、对数阶 $O(\log_2 n)$、线形阶 $O(n)$、线形对数阶 $O(n\ \log_2 n)$、平方阶 $O(n^3)$、…、k 次方阶 $O(n^k)$、指数阶 $O(2^n)$。显然，时间复杂度为指数阶 $O(2^n)$的算法效率极低，当 n 值稍大时就无法应用。

2. 算法空间复杂度

类似于时间复杂度，一个算法的空间复杂度（space complexity）$S(n)$ 定义为该算法所耗费的存储空间，它也是问题规模 n 的函数。渐近空间复杂度也常常简称为空间复杂度。算法的时间复杂度和空间复杂度合称为算法的复杂度。

1．2．3　常用的算法

(1)列举法

就是枚举法，适合项目少的情况。数量太多，无法列举。

(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，最后找出一般的关系。显然，归纳法要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。但是，从一个实际问题中总结归纳出一般的关系，并不是一件容易的事情，尤其是要归纳出一个数学模型更为困难。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。

归纳是一种抽象，即从特殊现象中找出一般关系。但由于在归纳的过程中不可能对所有的情况进行列举，因此，最后由归纳得到的结论还只是一种猜测，还需要对这种猜测加以必要的证明。实际上，通过精心观察而得到的猜测得不到证实或最后证明猜测是错的，也是常有的事。

(3)递推

所谓递推，是指从已知的初始条件出发，逐次推出所要求的各中间结果和最后结果。其中初始条件或是问题本身已经给定，或是通过对问题的分析与化简而确定。递推本质上也属于归纳法，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，递推关系式往往是归纳的结果。

递推算法在数值计算中是极为常见的。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

人们在解决一些复杂问题时，为了降低问题的复杂程度(如问题的规模等)，一般总是

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，实际上并没有对问题进行求解，而只是当解决了最后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，这就是递归的基本思想。由此可以看出，递归的基础也是归纳。在工程实际中，有许多问题就是用递归来定义的，数学中的许多函数也是用递归来定义的。递归在可计算性理论和算法设计中占有很重要的地位。

递归分为直接递归与间接递归两种。如果一个算法P显式地调用自己则称为直接递归。如果算法 P 调用另一个算法Q，而算法Q又调用算法P，则称为间接递归调用。

递归是很重要的算法设计方法之一。实际上，递归过程能将一个复杂的问题归结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，直到归结为最简单的问题为止。

有些实际问题，既可以归纳为递推算法，又可以归纳为递归算法。但递推与递归的实现方法是大不一样的。递推是从初始条件出发，逐次推出所需求的结果；而递归则是从算法本身到达递归边界的。通常，递归算法要比递推算法清晰易读，其结构比较简练。特别是在许多比较复杂的问题中，很难找到从初始条件推出所需结果的全过程，此时，设计递归算法要比递推算法容易得多。但递归算法的执行效率比较低。

(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”，是指将问题的规模减半，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

前面讨论的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，有些实际问题很难归纳出一组简单的递推公式或直观的求解步骤，并且也不能进行无限的列举。对于这类问题，一种有效的方法是“试”。通过对问题的分析，找出一个解决问题的线索，然后沿着这个线索逐步试探，对于每一步的试探，若试探成功，就得到问题的解，若试探失败，就逐步回退，换别的路线再进行试探。这种方法称为回溯法。回溯法在处理复杂数据结构方面有着广泛的应用。

1.3 线性表

线性结构是最简单、最常用的数据结构。

1．3．1 线性表的定义

线性表（Linear list）是由n（$n\geq 0$）个数据元素（结点）a_1，a_2，…，a_n组成的有限序

列。线性表中数据元素之间的逻辑关系是：对表中任一个结点，与它相邻且在它前面的结点（亦称为直接前驱（immediate predecessor）最多只有一个；与表中的任一结点相邻且在其后的结点（亦称为直接后继（immediate successor）也最多只有一个。表中只有第一个结点没有直接前驱，故称为开始结点；也只有最后一个结点没有直接后继，故称之为终端结点。

例如一年四季是一个长度为 4 的线形表，表中的每一个季节名就是一个数据元素。英文字母表（*A*，*B*，…，*Z*）是线性表，表中每个字母是一个数据元素（结点）。一副扑克牌的点数（2，3，…，10，*J*，*Q*，*K*，*A*）也是一个线性表，其中数据元素是每张牌的点数。

线性表具有下列的逻辑特征：

（1）数据元素的个数 *n* 定义为表的长度（*n*=0 时称为空表）。

（2）将非空的线性表（*n*>0）记作：（*a*1，*a*2，…，*an*）。

（3）数据元素 a_i（1≤*i*≤*n*）只是个抽象符号，其具体含义在不同情况下可以不同。

1．3．2　线性表的顺序存储结构

在计算机中存储线性表，若按顺序存储的话，这时的线性表就是顺序表。

1．顺序表的定义

即把线性表的结点按逻辑次序依次存放在一组地址连续的存储单元里的方法，称为顺序存储方法。

用顺序存储方法存储的线性表称为顺序表（sequential list）。

2．顺序表的特点

顺序表是用向量实现的线性表，向量的下标可以看作是结点的相对地址。因此，顺序表的特点是逻辑上相邻的结点，其物理位置亦相邻。

3．结点 *ai* 的存储地址

不失一般性，设线性表中所有结点的类型相同，则每个结点所占用存储空间大小亦相同。假设表中每个结点占用 *c* 个存储单元，其中第一个单元的存储地址则是该结点的存储地址，并设表中开始记结点的存储地址（简称为基地址）是 $LOC(a_1)$，那么结点 a_i 的存储地址 LOC（*ai*）可通过下式计算：

$$LOC(a_i)=LOC(a_1)+(i-1)*c \quad 1\le i\le n$$

在顺序表中，每个结点 a_i 的存储地址是该结点在表中的位置 *i*≤1 的线性函数。只要知道基地址和每个结点的大小，就可在相同时间内求出任一结点的存储地址。因此，线性表是一种随机存取。

在计算机中顺序存储结构如图 1－2 所示。

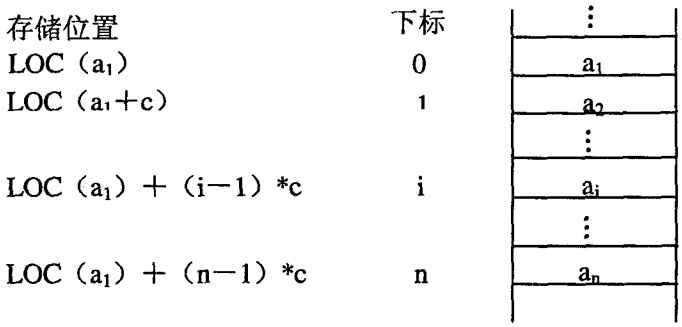

图 1−2 顺序表存储结构

4．顺序表类型定义

```
#define ListSize 100//表空间的大小可根据实际需要而定，这里假设为 100
typedef int DataType;//DataeType 的类型可根据实际情况而定，这里假设为 int
typedef struct{
    DataeType data[ListSize];//data 用于存放表结点
    int length;//当前的表长度
}SeqList;
```

除用向量这种顺序存储的数组类型存储形表的元素外，顺序表还应该用一个变量来表示线性表的长度属性，因此用结构类型来定义顺序表类型。存放线性表结点的向量空间的大小 ListSize 应仔细选值，使其既能满足表结点的数目动态增加的需求，又不至于预先定义过大而浪费存储空间。在 C 语言中向量的下标从 0 开始，所以若 L 是 SeqList 类型的顺序表，在线性表的考试结点 a_1 和终端结点 a_n 分别存储在 L.data[0]和 L.data[l.length-1]中。

1．3．3 线性表的插入运算

1．插入运算的逻辑描述

线性表的插入运算是指在表的第 i（$1 \le i \le n+1$）个位置上，插入一个新结点 x，使长度为 n 的线性表（a_1，…，a_{i-1}，a_i，…a_n）变成长度为 $n+1$ 的线性表（a_1，…，a_{i-1}，x，a_i，…a_n）。

由于向量空间大小在声明时确定，当 L−>length≥ListSize 时，表空间已满，不可再做插入操作。当插入位置 i 的值为 $i>n$ 和 $i<1$ 时为非法位置，不可做正常插入操作。

2．顺序表插入操作过程

在顺序表中，结点的物理顺序必须和结点的逻辑顺序保持一致，因此必须将表中位置为 n，$n-1$，…，i 上的结点，依次后移到位置 $n+1$，n，…，$i+1$ 上，空出第 i 个位置，然后在该位置上插入新结点 x，并将表长加 1。仅当插入位置 $i=n+1$ 时，才无须移动结点，