全国计算机等级考试辅导用书

飞思考试中心

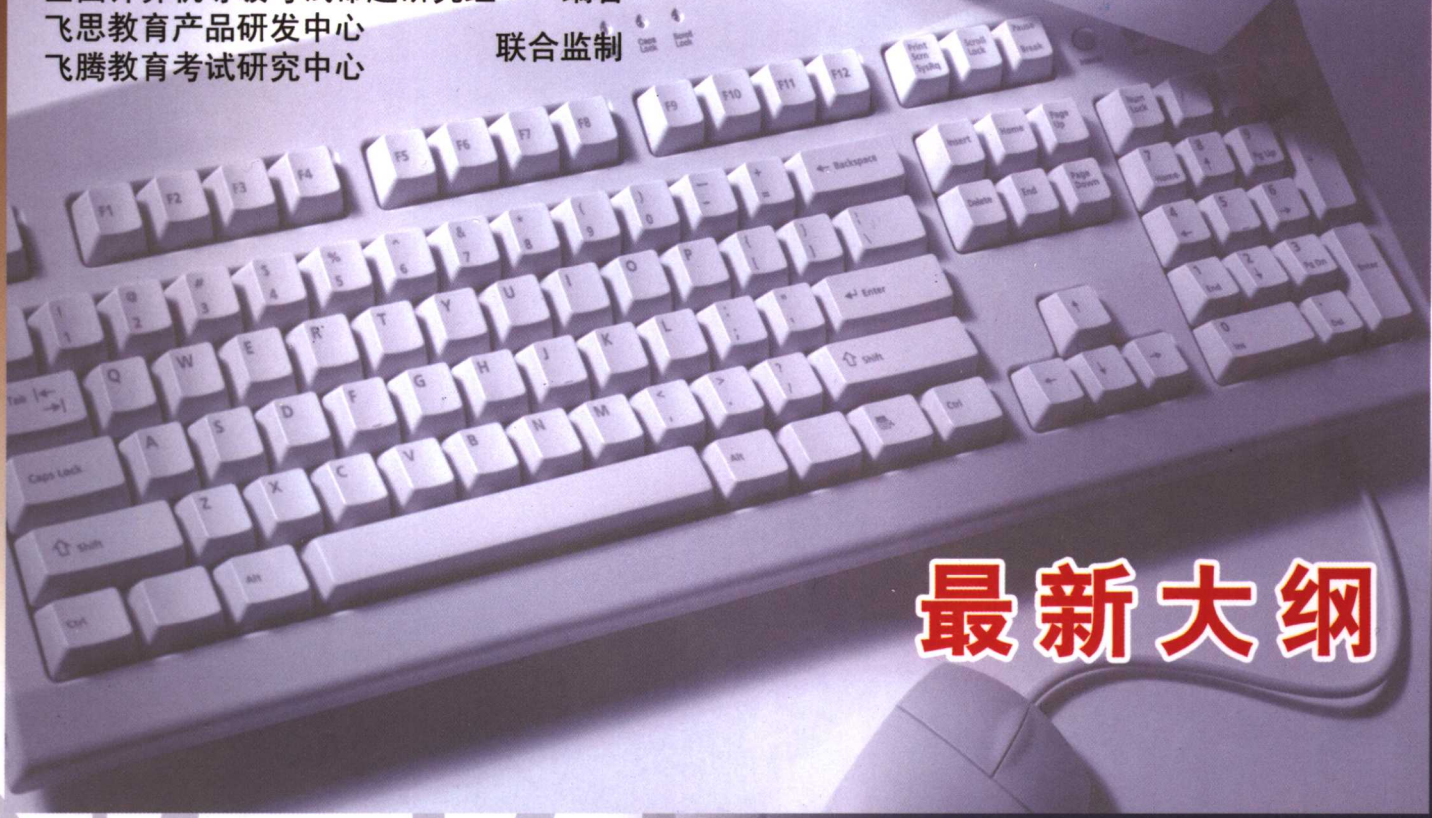
National Computer Rank Examination

全国计算机等级考试

考点分析、题解与模拟

(二级公共基础知识)

全国计算机等级考试命题研究组 编著
飞思教育产品研发中心
飞腾教育考试研究中心 联合监制



最新大纲

NCRE

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

飞思考试中心

全国计算机等级考试考点分析、题解与模拟

（二级公共基础知识）

全国计算机等级考试命题研究组　编著
飞思教育产品研发中心
飞腾教育考试研究中心　联合监制

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内 容 简 介

本书依据教育部考试中心最新发布的2004年版《全国计算机等级考试考试大纲》编写而成，一方面结合命题规律，对重要考点进行分析、讲解，并选取经典考题深入剖析；另一方面配有同步练习、模拟试题和上机试题，逐步向考生详尽透析考试中的所有知识要点。可谓“一书在手，通关无忧”。

本书适合于作为全国计算机等级考试考前培训班辅导用书，也可作为应试人员的自学用书。

图书在版编目(CIP)数据

全国计算机等级考试考点分析、题解与模拟(二级公共基础知识)/全国计算机等级考试命题研究组编著. —北京：电子工业出版社，2005.1
(飞思考试中心)
ISBN 7-121-00709-6

Ⅰ.全… Ⅱ.全… Ⅲ.电子计算机—水平考试—自学参考资料 Ⅳ.TP3

中国版本图书馆CIP数据核字(2004)第132923号

责任编辑：杨 鸽
印 刷：北京中科印刷有限公司
出版发行：电子工业出版社
北京海淀区万寿路173信箱 邮编：100036
经 销：各地新华书店
开 本：880×1230 1/16 印张：17 字数：489.6千字
印 次：2005年1月第1次印刷
定 价：19.80元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系电话：010-68279077。质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

Preface 前言

全国计算机等级考试自1994年由国家教育部考试中心推出以来，其评测全社会的非计算机专业人员的计算机知识与技能，为培养各行业的计算机应用人才开辟了一条新的道路，也受到用人单位和学习人员的热烈欢迎。全国计算机等级考试通过数年的发展，已经成为我国最大型的计算机类考试。

为了帮助更多的学习者顺利地通过考试，并掌握相应的操作技能，我们在深入调研、详尽分析考试大纲的基础上，组织国内著名高校的计算机专家和一线教师编写了本书。

本书分为四大部分。

※ 考点分析

“考点分析”结合2004年版最新考试大纲、教材，对教材中考核的重点和难点进行讲解，涵盖了大纲中所有考试的考核点。

※ 经典题解

“经典题解”选取极具代表性的经典例题，例题符合考试命题规律的特征，对题目的讲解深入、透彻，循序渐进，条理性强。

※ 同步练习

“同步练习”提供了大量习题，对前面所学的理论知识进行温习和巩固，以练促学、学练结合。

※ 全真模拟试题

结合最新考试大纲，筛选与演绎出的典型试题集，不论是形式上还是难度上都与真题类似，解析详尽、透彻。

由于时间仓促，书中难免有不当之处，敬请指正。

我们的联系方式：
电　　话：(010)68134545　68131648　62754774
电子邮件：support@fecit.com.cn　eduexam@vip.sina.com
飞思在线：http://www.fecit.com.cn　http://www.fecit.net
中国教育考试网：http://www.eduexam.cn
通用网址：计算机图书、飞思、飞思教育、飞思科技、FECIT

全国计算机等级考试命题研究组
飞思教育产品研发中心

Contents 目录

第 1 章　数据结构与算法

1.1　算法 …… 2
1.2　数据结构的基本概念 …… 4
1.3　线性表及顺序存储结构 …… 6
1.4　栈和队列 …… 9
1.5　线性链表 …… 12
1.6　树与二叉树 …… 15
1.7　查找技术 …… 19
1.8　排序技术 …… 20
1.9　经典解析 …… 24
1.10　同步练习 …… 54
1.11　参考答案 …… 66

第 2 章　程序设计基础

2.1　程序设计方法与风格 …… 70
2.2　结构化程序设计 …… 71
2.3　面向对象的程序设计 …… 73
2.4　经典解析 …… 76
2.5　同步练习 …… 85
2.6　参考答案 …… 89

第 3 章　软件工程基础

3.1　软件工程基本概念 …… 92
3.2　结构化分析方法 …… 96
3.3　结构化设计方法 …… 99
3.4　软件的测试 …… 106
3.5　程序的调试 …… 110
3.6　软件工程管理 …… 111
3.7　经典解析 …… 112
3.8　同步练习 …… 139
3.9　参考答案 …… 158

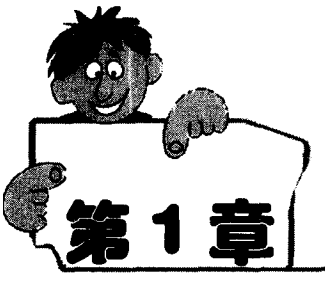
第 4 章　数据库设计基础

4.1　数据库系统的基本概念 …… 164
4.2　数据模型 …… 170

4.3 关系代数 …… 177
4.4 数据库的设计与管理 …… 179
4.5 经典解析 …… 187
4.6 同步练习 …… 210
4.7 参考答案 …… 229

第5章 笔试全真模拟试题

5.1 笔试全真模拟试题(1) …… 234
5.2 笔试全真模拟试题(2) …… 235
5.3 笔试全真模拟试题(3) …… 236
5.4 笔试全真模拟试题(4) …… 237
5.5 笔试全真模拟试题(5) …… 238
5.6 笔试全真模拟试题(6) …… 239
5.7 笔试全真模拟试题(7) …… 240
5.8 笔试全真模拟试题(8) …… 241
5.9 笔试全真模拟试题(9) …… 242
5.10 笔试全真模拟试题(10) …… 243
5.11 笔试全真模拟试题(11) …… 244
5.12 笔试全真模拟试题(12) …… 244
5.13 笔试全真模拟试题(13) …… 245
5.14 参考答案及解析 …… 246



第1章 数据结构与算法

考核知识点

- 算法的基本概念。
- 数据结构的基本概念。
- 线性表的定义。
- 栈和队列。
- 线性链表。
- 循环链表。
- 树的基本概念。
- 查找技术。
- 排序技术。

重要考点提示

- 算法的基本概念;算法复杂度的概念和意义(时间复杂度与空间复杂度)。
- 数据结构的定义;数据的逻辑结构与存储结构;数据结构的图形表示;线性结构与非线性结构。
- 线性表的定义;线性表的顺序存储结构及其插入与删除运算。
- 栈和队列的定义;栈和队列的顺序存储结构及其基本运算。
- 线性单链表、双向链表与循环链表的结构及其基本运算。
- 树的基本概念;二叉树的定义及其存储结构;二叉树的前序、中序和后序遍历。
- 顺序查找与二分法查找算法。
- 基本排序算法(交换类排序、选择类排序、插入类排序)。

1.1 算　法

考点1 算法的基本概念

计算机解题的过程实际上是在实施某种算法，这种算法称为计算机算法。

算法（algorithm）是一组严谨地定义运算顺序的规则，并且每一个规则都是有效的，同时是明确的，此顺序将在有限的次数后终止。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，其中每一条指令表示一个或多个操作。

1. 算法的基本特征

（1）可行性（effectiveness）：针对实际问题而设计的算法，执行后能够得到满意的结果。

（2）确定性（definiteness）：算法的中的每一个步骤都必须有明确的定义，不允许有模棱两可的解释和多义性。

（3）有穷性（finiteness）：算法必须在有限时间内做完，即算法必须能在执行有限个步骤之后终止。

（4）拥有足够的情报：要使算法有效必须为算法提供足够的情报。当算法拥有足够的情报时，此算法才是有效的；而当提供的情报不够时，算法可能无效。

2. 算法的基本要素

（1）算法中对数据的运算和操作：每个算法实际上是按解题要求从环境能进行的所有操作中选择合适的操作所组成的一组指令序列。

计算机可以执行的基本操作是以指令的形式描述的。一个计算机系统能执行的所有指令的集合，称为该计算机系统的指令系统。计算机程序就是按解题要求从计算机指令系统中选择合适的指令所组成的指令序列。在一般的计算机系统中，基本的运算和操作有以下4类：

①算术运算：主要包括加、减、乘、除等运算；

②逻辑运算：主要包括“与”、“或”、“非”等运算；

③关系运算：主要包括“大于”、“小于”、“等于”、“不等于”等运算；

④数据传输：主要包括赋值、输入、输出等操作。

（2）算法的控制结构：一个算法的功能不仅仅取决于所选用的操作，而且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。

算法的控制结构给出了算法的基本框架，它不仅决定了算法中各操作的执行顺序，而且也直接反映了算法的设计是否符合结构化原则。描述算法的工具通常有传统流程图、N－S结构化流程图、算法描述语言等。一个算法一般都可以用顺序、选择、循环3种基本控制结构组合而成。

3. 算法设计的基本方法

计算机算法不同于人工处理的方法，下面是工程上常用的几种算法设计，在实际应用时，各种方法之间往往存在着一定的联系。

（1）列举法

列举法是计算机算法中的一个基础算法。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检验哪些是需要的，哪些是不需要的。

列举法的特点是算法比较简单。但当列举的可能情况较多时，执行列举算法的工作量将会很大。因此，在用列举法设计算法时，使方案优化，尽量减少运算工作量，是应该重点注意的。

（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。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

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。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，再沿着原来的分解的逆过程逐步进行综合，这就是递归的基本思想。

递归分为直接递归与间接递归两种。

(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。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

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，若试探成功，就得到问题的解，若试探失败，就逐步回退，换别的路线再逐步试探。

4. 算法设计的要求

通常一个好的算法应达到如下目标：

(1)正确性(correctness)

正确性大体可以分为以下4个层次：

①程序不含语法错误；

②程序对于几组输入数据能够得出满足规格说明要求的结果；

③程序对于精心选择的典型、苛刻而带有刁难性的几组输入数据能够得出满足规格说明要求的结果；

④程序对于一切合法的输入数据都能产生满足规格说明要求的结果。

(2)可读性(readability)

算法主要是为了方便人的阅读与交流，其次才是其执行。可读性好有助于用户对算法的理解；晦涩难懂的程序易于隐藏较多错误，难以调试和修改。

(3)健壮性(robustness)

当输入数据非法时，算法也能适当地做出反应或进行处理，而不会产生莫名其妙的输出结果。

(4)效率与低存储量需求

效率指的是程序执行时，对于同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过程中所需要的最大存储空间。

考点2 算法的复杂度

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。同一个算法用不同的语言实现，或者用不同的编译程序进行编译，或者在不同的计算机上运行，效率均不同。这表明使用绝对的时间单位衡量算法的效率是不合适的。撇开这些与计算机硬件、软件有关的因素，可以认为一个特定算法“运行工作量”的大小，

只依赖于问题的规模(通常用整数 n 表示),它是问题的规模函数。即

$$\text{算法的工作量} = f(n)$$

例如,在 $N \times N$ 矩阵相乘的算法中,整个算法的执行时间与该基本操作(乘法)重复执行的次数 n^3 成正比,也就是时间复杂度为 n^3,即

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

在有的情况下,算法中的基本操作重复执行的次数还随问题的输入数据集不同而不同。例如在起泡排序的算法中,当要排序的数组 a 初始序列为自小至大有序时,基本操作的执行次数为 0;当初始序列为自大至小有序时,基本操作的执行次数为 $n(n-1)/2$。对这类算法的分析,可以采用以下两种方法来分析。

(1)平均性态(Average Behavior)

所谓平均性态是指各种特定输入下的基本运算次数的加权平均值来度量算法的工作量。

设 x 是所有可能输入中的某个特定输入,$p(x)$ 是 x 出现的概率(即输入为 x 的概率),$t(x)$ 是算法在输入为 x 时所执行的基本运算次数,则算法的平均性态定义为

$$A(n) = \sum_{x \in Dn} P(x)t(x)$$

其中 Dn 表示当规模为 n 时,算法执行的所有可能输入的集合。

(2)最坏情况复杂性(Worst-case Complexity)

所谓最坏情况分析,是指在规模为 n 时,算法所执行的基本运算的最大次数。

$$W(n) = \max_{x \in Dn}\{t(x)\}$$

2. 算法的空间复杂度

算法的空间复杂度是指执行这个算法所需要的内存空间。

一个算法所占用的存储空间包括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。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(in place)工作的。在许多实际问题中,为了减少算法所占的存储空间,通常采用压缩存储技术,以便尽量减少不必要的额外空间。

1.2 数据结构的基本概念

考点 3 数据结构的定义

数据结构(data structure)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,即数据的组织形式。

数据结构作为计算机的一门学科,主要研究和讨论以下三个方面:

(1)数据集合中个数据元素之间所固有的逻辑关系,即数据的逻辑结构;

(2)在对数据元素进行处理时,各数据元素在计算机中的存储关系,即数据的存储结构;

(3)对各种数据结构进行的运算。

讨论以上问题的目的是为了提高数据处理的效率,所谓提高数据处理的效率有两个方面:

(1)提高数据处理的速度;

(2)尽量节省在数据处理过程中所占用的计算机存储空间。

数据(data):是对客观事物的符号表示,在计算机科学中是指所有能输入到计算机中并被计算机程序处理的符号的总称。

数据元素(data element):是数据的基本单位,在计算机程序中通常作为一个整体进行考虑和处理。

数据对象(data object):是性质相同的数据元素的集合,是数据的一个子集。

在一般情况下,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有某种关系(即连续),这种关系反映了该集合中的数据元素所固有的一种结构。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。

前后件关系是数据元素之间的一个基本关系,但前后件关系所表示的实际意义随具体对象的不同而不同。一般来说,数据元素之间的任何关系都可以用前后件关系来描述。

1. 数据的逻辑结构

数据结构是指反映数据元素之间的关系的数据元素集合的表示。更通俗地说,数据结构是指带有结构的数据元素的集合。所谓结构实际上就是指数据元素之间的前后件关系。

一个数据结构应包含以下两方面信息:

(1)表示数据元素的信息;

(2)表示各数据元素之间的前后件关系。

数据的逻辑结构是对数据元素之间的逻辑关系的描述,它可以用一个数据元素的集合和定义在此集合中的若干关系来表示。

数据的逻辑结构包括集合、线性结构、树型结构和图形结构四种。

线性结构:数据元素之间构成一种顺序的线性关系。如图 1-1 所示。

图 1-1 线性结构

树型结构:数据元素之间形成一种树型的关系。如图 1-2 所示。

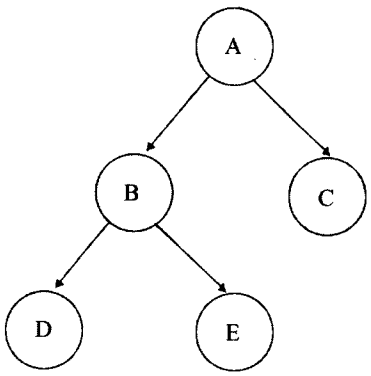


图 1-2 树型结构

数据的逻辑结构有两个要素:一是数据元素的集合,通常记为 D;二是 D 上的关系,它反映了数据元素之间的前后件关系,通常记为 R。一个数据结构可以表示成

$$B = (D, R)$$

其中 B 表示数据结构。为了反映 D 中各数据元素之间的前后件关系,一般用二元组来表示。

例如,复数是一种数据结构,在计算机科学中,复数可取如下定义:

$$B = (C, R)$$

其中,C 是含有两个实数的集合$\{c1, c2\}$;R 是定义在集合 C 上的一种关系$\{<c1, c2>\}$,其中有序偶$\{<c1, c2>\}$表示 $c1$ 是复数的实部,$c2$ 是复数的虚部。

2. 数据的存储结构

数据的逻辑结构在计算机存储空间中的存放形式称为数据的存储结构(也称数据的物理结构)。

由于数据元素在计算机存储空间中的位置关系可能与逻辑关系不同,因此,为了表示存放在计算机存储空间中的各数据元素之间的逻辑关系(即前后件关系),在数据的存储结构中,不仅要存放各数据元素的信息,还需要存放各数据元素之间的前后件关系的信息。

一种数据的逻辑结构根据需要可以表示成多种存储结构，常用的存储结构有顺序、链接、索引等存储结构。而采用不同的存储结构，其数据处理的效率是不同的。因此，在进行数据处理时，选择合适的存储结构是很重要的。

考点4 数据结构的图形表示

数据结构除了用二元关系表示外，还可以直观地用图形表示。

在数据结构的图形表示中，对于数据集合 D 中的每一个数据元素用中间标有元素值的方框表示，一般称之为数据结点，并简称为结点；为了进一步表示各数据元素之间的前后件关系，对于关系 R 中的每一个二元组，用一条有向线段从前件结点指向后件结点。

例如，反映家庭成员间辈分关系的数据结构可以用如图 1-3 所示。

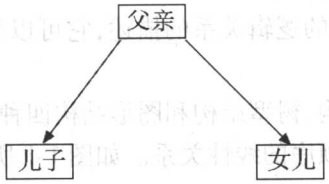


图 1-3 家庭成员辈分关系数据结构的图形表示

在数据结构中，没有前件的结点称为根结点；没有后件的结点称为终端结点（也称为叶子结点）。

一个数据结构中的结点可能是在动态变化的。根据需要或在处理过程中，可以在一个数据结构中增加一个新结点（称为插入运算），也可以删除数据结构中的某个结点（称为删除运算）。插入与删除是对数据结构的两种基本运算。除此之外，对数据结构的运算还有查找、分类、合并、分解、复制和修改等。

考点5 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。

根据数据结构中各数据元素之间前后件关系的复杂程度，一般将数据结构分为两大类型：线性结构与非线性结构。

非空数据结构满足：

（1）有且只有一个根结点；

（2）每一个结点最多有一个前件，也最多有一个后件。

则称该数据结构为线性结构。线性结构又称为线性表。一个线性表是 n 个数据元素的有限序列。至于每个元素的具体含义，在不同的情况下各不相同，它可以是一个数或一个符号，也可以是一页书，甚至其他更复杂的信息。如果一个数据结构不是线性结构，称之为非线性结构。线性结构与非线性结构都可以是空的数据结构。对于空的数据结构，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 线性表及顺序存储结构

考点6 线性表的定义

线性表是 $n(n\geqslant 0)$ 个元素构成的有限序列 $(a_1, a_2, \cdots, a_n)$。表中的每一个数据元素，除了第一个外，有

且只有一个前件，除了最后一个外，有且只有一个后件。即线性表是一个空表，或可以表示为

$$(a_1, a_2, \cdots, a_n)$$

其中 $a_i(i=1,2,\cdots,n)$ 是属于数据对象的元素，通常也称其为线性表中的一个结点。

其中，每个元素可以简单到是一个字母或是一个数据，也可能是比较复杂的由多个数据项组成的。在复杂的线性表中，由若干数据项组成的数据元素称为记录(record)，而由多个记录构成的线性表又称为文件(file)。在非空表中的每个数据元素都有一个确定的位置，如 a_1 是第一个元素，a_n 是最后一个数据元素，a_i 是第 i 个数据元素，称 i 为数据元素 a_i 在线性表中的位序。非空线性表有如下一些结构特征：

(1)有且只有一个根结点 a_1，它无前件；

(2)有且只有一个终端结点 a_n，它无后件；

(3)除根结点与终端结点外，其他所有结点有且只有一个前件，也有且只有一个后件。线性表中结点的个数 n 称为线性表的长度。当 $n=0$ 时称为空表。

考点7 线性表的顺序存储结构

线性表的顺序表指的是用一组地址连续的存储单元依次存储线性表的数据元素。

线性表的顺序存储结构具备如下两个基本特征：

(1)线性表中的所有元素所占的存储空间是连续的；

(2)线性表中各数据元素在存储空间中是按逻辑顺序依次存放的。

假设线性表的每个元素需要占用 k 个存储单元，并以所占的存储位置 $ADR(a_{i+1})$ 和第 i 个数据元素的存储位置 $ADR(a_i)$ 之间满足下列关系：

$$ADR(a_{i+1}) = ADR(a_i) + k$$

线性表第 i 个元素 a_i 的存储位置为

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

式中 $ADR(a_i)$ 是线性表的第一个数据元素 a_1 的存储位置，通常称做线性表的起始位置或基址。

线性表的这种表示称做线性表的顺序存储结构或顺序映像，这种存储结构的线性表为顺序表。表中每一个元素的存储位置都和线性表的起始位置相差一个和数据元素在线性表中的位序成正比例的常数。如图1-4所示。由此只要确定了存储线性表的起始位置，线性表中任一数据元素都可以随机存取，所以线性表的顺序存储结构是一种随机存取的存储结构。

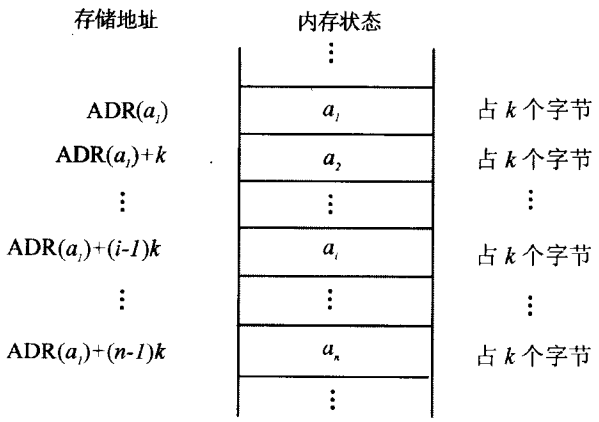


图1-4 线性表顺序存储结构示意图

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。在线性表的顺序存储结构下，可以对线性表做以下运算：

(1)在线性表的指定位置处加入一个新的元素(即线性表的插入)；
(2)在线性表中删除指定的元素(即线性表的删除)；
(3)在线性表中查找某个(或某些)特定的元素(即线性表的查找)；
(4)对线性表中的元素进行整序(即线性表的排序)；
(5)按要求将一个线性表分解成多个线性表(即线性表的分解)；
(6)按要求将多个线性表合并成一个线性表(即线性表的合并)；
(7)复制一个线性表(即线性表的复制)；
(8)逆转一个线性表(即线性表的逆转)等。

考点8 顺序表的插入运算

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

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

变成长度为 $n+1$ 的线性表

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

现在分析算法的复杂度。这里的问题规模是表的长度，设它的值为 n。该算法的时间主要花费在循环的结点后移语句上，该语句的执行次数(即移动结点的次数)是 $n-i+1$。由此可看出，所需移动结点的次数不仅依赖于表的长度，而且还与插入位置有关。

当 $i=n+1$ 时，由于循环变量的终值大于初值，结点后移语句将不进行；这是最好情况，其时间复杂度 $O(1)$；

当 $i=1$ 时，结点后移语句，将循环执行 n 次，需移动表中所有结点，这是最坏情况，其时间复杂度为 $O(n)$。

由于插入可能在表中任何位置上进行，因此需分析算法的平均复杂度。

在长度为 n 的线性表中第 i 个位置上插入一个结点，令 $Eis(n)$ 表示移动结点的期望值(即移动的平均次数)，则在第 i 个位置上插入一个结点的移动次数为 $n-i+1$。故

$$Eis(n)=\sum_{i=1}^{n+1}p_i(n-i+1)$$

不失一般性，假设在表中任何位置 $(1\leqslant i\leqslant n+1)$ 上插入结点的机会是均等的，则

$$p_1=p_2=p_3=\cdots=p_{n+1}=1/(n+1)$$

因此，在等概率插入的情况下，

$$Eis(n)=\sum_{i=1}^{n+1}p_i(n-i+1)/(n+1)=n/2$$

也就是说，在顺序表上做插入运算，平均要移动表上一半的结点。当表长 n 较大时，算法的效率相当低。虽然 $Eis(n)$ 中 n 的的系数较小，但就数量级而言，它仍然是线性级的。因此算法的平均时间复杂度为 $O(n)$。如图 1-5 所示。

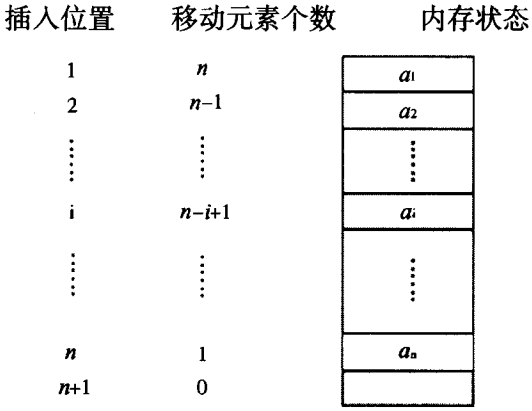


图 1-5 插入操作时间复杂度示意图

考点9 顺序表的删除运算

线性表的删除运算是指将表的第 $i(1 \leqslant i \leqslant n)$ 个结点删除，使长度为 n 的线性表：

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

变成长度为 $n-1$ 的线性表

$$(a_1, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n)$$

该算法的时间分析与插入算法相似，结点的移动次数也是由表长 n 和位置 i 决定。若 $i=n$，则由于循环变量的初值大于终值，前移语句将不执行，无需移动结点；若 $i=1$，则前移语句将循环执行 $n-1$ 次，需移动表中除开始结点外的所有结点。这两种情况下算法的时间复杂度分别为 $O(1)$ 和 $O(n)$。

删除算法的平均性能分析与插入算法相似。在长度为 n 的线性表中删除一个结点，令 $Ede(n)$ 表示所需移动结点的平均次数，删除表中第 i 个结点的移动次数为 $n-i$，故

$$Ede(n) = \sum_{i=1}^{n} p_i(n-i)$$

式子中，pi 表示删除表中第 i 个结点的概率。在等概率的假设下，

$$p_1 = p_2 = p_3 = \cdots = p_n = 1/n$$

由此可得：

$$Ede(n) = \sum_{i=1}^{n} p_i(n-i)/n = (n-1)/2$$

即在顺序表上做删除运算，平均要移动表中约一半的结点，平均时间复杂度也是 $O(n)$。

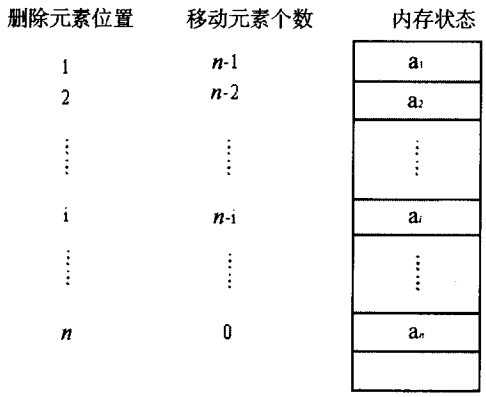


图 1-6 删除操作时间复杂度示意图

1.4 栈和队列

考点10 栈及其基本运算

1. 什么是栈

栈实际也是线性表，只不过是一种特殊的线性表。栈(Stack)是只能在表的一端进行插入和删除运算的线性表，通常称插入、删除的这一端为栈顶(Top)，另一端为栈底(Bottom)。当表中没有元素时称为空栈。栈顶元素总是后被插入的元素，从而也是最先被删除的元素；栈底元素总是最先被插入的元素，从而也是最后才能被删除的元素。

假设栈 $S=(a_1, a_2, a_3, \cdots, a_n)$，则 a_1 称为栈底元素，a_n 为栈顶元素。栈中元素按 $a_1, a_2, a_3, \cdots, a_n$ 的次

序进栈,退栈的第一个元素应为栈顶元素。换句话说,栈的修改是按后进先出的原则进行的。因此,栈称为先进后出表(FILO,First In Last Out)。或"后进先出"表(LIFO,Last In First Out)。如图 1-7 所示。

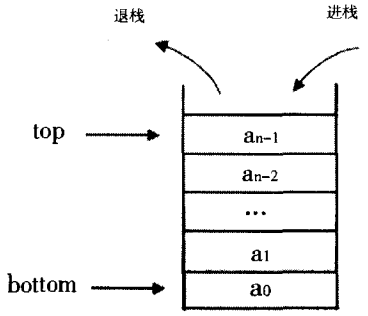


图 1-7 进栈出栈示意图

2. 栈的顺序存储及其运算

(1)入栈运算:入栈运算是指在栈顶位置插入一个新元素。首先将栈顶指针加一(即 top 加 1),然后将新元素插入到栈顶指针指向的位置。当栈顶指针已经指向存储空间的最后一个位置时,说明栈空间已满,不可能再进行入栈操作。这种情况称为栈"上溢"错误。如图 1-8 所示。

(2)退栈运算:退栈是指取出栈顶元素并赋给一个指定的变量。首先将栈顶元素(栈顶指针指向的元素)赋给一个指定的变量,然后将栈顶指针减一(即 top 减 1)。当栈顶指针为 0 时,说明栈空,不可进行退栈操作。这种情况称为栈的"下溢"错误。如图 1-8 所示。

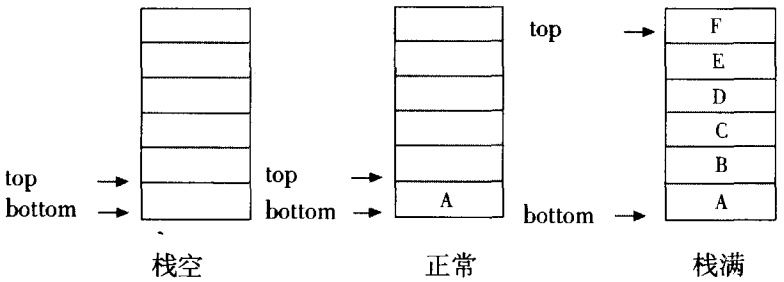


图 1-8 栈的状态

(3)读栈顶元素:读栈顶元素是指将栈顶元素赋给一个指定的变量。这个运算不删除栈顶元素,只是将它赋给一个变量,因此栈顶指针不会改变。当栈顶指针为 0 时,说明栈空,读不到栈顶元素。

考点 11 队列及其基本运算

1. 什么是队列

队列(queue)是只允许在一端删除,在另一端插入的顺序表,允许删除的一端叫做队头(front),允许插入的一端叫做队尾(rear)。如图 1-9 所示。

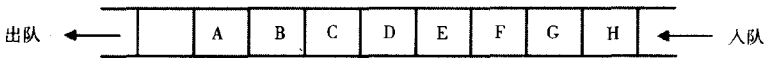
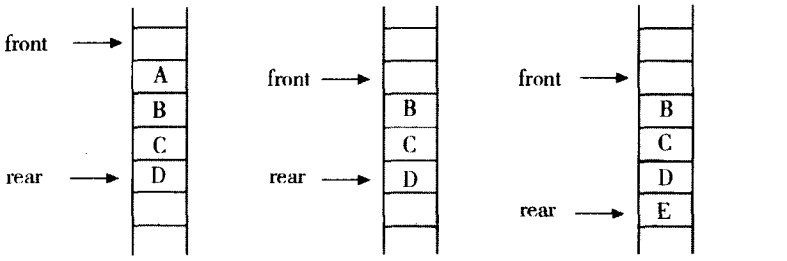


图 1-9 队列示意图

当队列中没有元素时称为空队列。在空队列中依次加入元素 $a_1,a_2,\cdots,a_n$ 之后,a_1 是队头元素,a_n 是队尾元素。显然退出队列的次序也只能是 $a_1,a_2,\cdots,a_n$,也就是说队列的修改是依先进先出的原则进行的。因此队列亦称作先进先出(FIFO,First In First Out)的线性表,或后进后出(LILO,Last In Last Out)的线性表。往队列队尾插入一个元素称为入队运算,从队列的排头删除一个元素称为退队运算。如图 1-10 所示。



(a)一个队列　(b)删除一个元素后的队列　(c)插入元素E后的队列

图 1-10　队列运算示意图

2. 循环队列及其运算

在实际应用中,队列的顺序存储结构一般采用循环队列的形式。所谓循环队列,就是将队列存储空间的最后一个位置绕到第一个位置,形成逻辑上的环状空间。如图 1-11 所示。

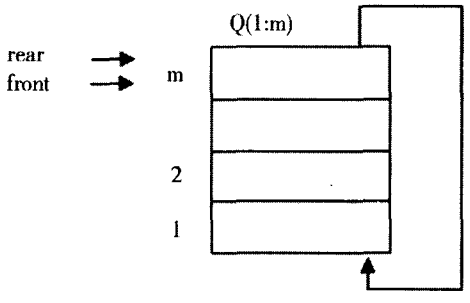


图 1-11　循环队列存储空间示意图

在循环队列中,用队尾指针 rear 指向队列中的队尾元素,用排头指针 front 指向排头元素的前一个位置,因此,从排头指针 front 指向的后一个位置直到队尾指针 rear 指向的位置之间所有的元素均为队列中的元素。

可以将向量空间想象为一个首尾相接的圆环,如图 1-12 所示,并称这种向量为循环向量,存储在其中的队列称为循环队列(Circular Queue)。在循环队列中进行出队、入队操作时,头尾指针仍要加 1,朝前移动。只不过当头尾指针指向向量上界(QueueSize - 1)时,其加 1 操作的结果是指向向量的下界 0。

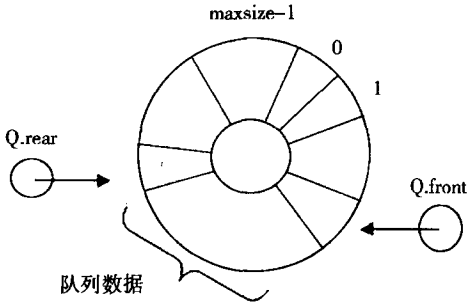


图 1-12　循环队列示意图

由于入队时尾指针向前追赶头指针,出队时头指针向前追赶尾指针,故队空和队满时头尾指针均相等。因此,我们无法通过 front = rear 来判断队列"空"还是"满"。

在实际使用循环队列时,为了能区分队列满还是队列空,通常还需增加一个标志 s,s 值的定义如下:当 $s=0$ 时表示队列空;当 $s=1$ 时表示队列非空。

(1)入队运算

入队运算是指在循环队列的队尾加入一个新元素。首先将队尾指针进一(即 rear = rear + 1),并当 rear $=m+1$ 时置 rear = 1;然后将新元素插入到队尾指针指向的位置。当循环队列非空($s=1$)且队尾指针等于队头指针时,说明循环队列已满,不能进行入队运算,这种情况称为"上溢"。