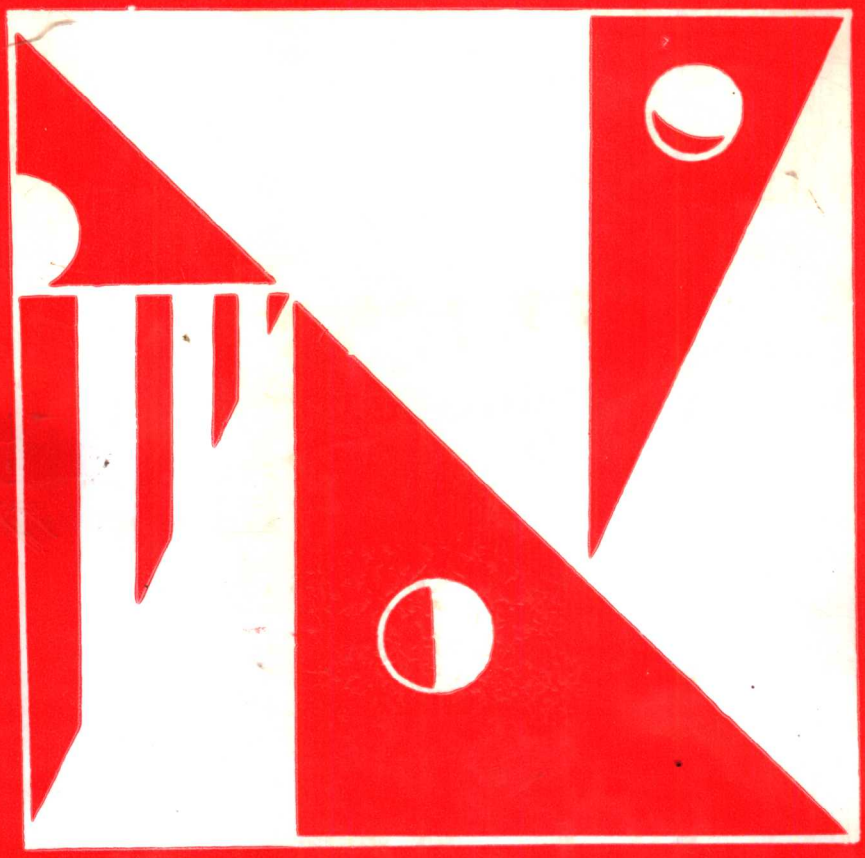
高等学校
工科电子类 规划教材

数据结构题例分析

朱静华　殷新春　编著



华中理工大学出版社

数据结构题例分析

朱静华　殷新春　编著

华中理工大学出版社

(鄂)新登字第 10 号

图书在版编目(CIP)数据

数据结构题例分析/朱静华 殷新春编著.
武汉:华中理工大学出版社,1995 年 8 月
ISBN 7-5609-1156-0
Ⅰ. 数…
Ⅱ. ①朱… ②殷…
Ⅲ. 程序设计-数据结构-算法分析
Ⅳ. TP311.12

数据结构题例分析
朱静华 殷新春 编著
责任编辑:沈旭日

华中理工大学出版社出版发行
武昌喻家山 邮编:430074)
新华书店湖北发行所经销
华中理工大学出版社照排室排版
武汉市新华印刷厂印刷
*
开本:787×1092 1/16 印张:14.25 字数:345 000
1995 年 8 月第 1 版 1995 年 8 月第 1 次印刷
印数:1—3 000
ISBN 7-5609-1156-0/TP·154
定价:8.40 元

内 容 提 要

本书扼要地介绍了数据结构的基本概念、逻辑特性和存储结构；对各种数据结构定义了相应的运算；通过大量的题例分析阐述了数据结构解题的思想与方法，以及在算法设计中如何根据实际问题及相应的操作选择合理的数据结构。

本书可作为计算机专业本科“数据结构”课程的教材或教学参考书，也可供广大数据处理工作者、计算机应用技术人员及自学者参考。

出 版 说 明

根据国务院关于高等学校教材工作的规定，我部承担了全国高等学校和中等专业学校工科电子类专业教材的编审、出版的组织工作。由于各有关院校及参与编审工作的广大教师共同努力，有关出版社的紧密配合，从1978～1990年，已编审、出版了三个轮次教材，及时供给高等学校和中等专业学校教学使用。

为了使工科电子类专业教材能更好地适应“三个面向”的需要，贯彻国家教委《高等教育“八五”期间教材建设规划纲要》的精神，“以全面提高教材质量水平为中心，保证重点教材，保持教材相对稳定，适当扩大教材品种，逐步完善教材配套”，作为“八五”期间工科电子类专业教材建设工作的指导思想，组织我部所属的八个高等学校教材编审委员会和四个中等专业学校专业教学指导委员会，在总结前三轮教材工作的基础上，根据教育形势的发展和教学改革的需要，制订了1991～1995年的“八五”（第四轮）教材编审出版规划。列入规划的，以主要专业主干课程教材及其辅助教材为主的教材约300余种。这批教材的评选推荐和编审工作，由各编委会或教学指导委员会组织进行。

这批教材的书稿，其一是从通过教学实践、师生反应较好的讲义中经院校推荐，由编审委员会（小组）评选择优产生出来的；其二是在认真遴选主编人的条件下进行约编的；其三是经过质量调查在前几轮组织编写出版的教材中修编的。广大编审者、各编审委员会（小组）、教学指导委员会和有关出版社，为保证教材的出版和提高教材的质量，作出了不懈的努力。

限于水平和经验，这批教材的编审、出版工作还可能有缺点和不足之处，希望使用教材的单位，广大教师和同学积极提出批评和建议，共同为不断提高工科电子类专业教材的质量而努力。

电子工业部教材办公室

前　言

随着计算机科学和技术的发展，计算机的功能不断增强，运算速度不断提高，应用于信息处理的范围日趋扩大，同时，计算机加工处理的对象也从简单的数值发展到一般的符号等，进而数据结构也相应地由简单变得更加复杂。

数据结构是计算机科学和技术中的一门技术基础课程，是设计和实现编译程序、操作系统、数据库系统及其它软件系统的重要基础。

数据结构的表示和操作都要涉及到算法。本书用类 PASCAL 语言作为算法表示的工具，因为这种语言不仅能较好地体现结构程序设计原则，而且还可以利用它丰富的数据类型来描述数据结构。使用类 PASCAL 语言表示算法时，可以集中精力于算法的本质，而不必考虑标准 PASCAL 语言中繁杂的类型、变量定义等说明成分。

全书共分十一章。第一章是数据结构与算法的基本概念，第二章至第五章介绍了数组、线性表、栈和队列、串等线性数据结构，第六章至第九章介绍了广义表、树、图、文件等非线性数据结构。这九章的内容提要部分扼要地介绍了各种数据结构的基本概念、逻辑特性和存储结构，并对每种数据结构定义了相应的运算。最后两章重点介绍了查找与内排序技术。

每章除内容提要外，主要通过大量题例分析来达到如下三个目的：①帮助理解“数据结构”课程的内容，强化基本概念；②训练程序设计技能，培养良好的设计风格；③正确掌握简单应用合理选择数据结构的方法。这些例题大多选自历年来各高校的硕士研究生入学试题、本科生考试试题。对一些典型的问题，给出了多种算法，并进行了详尽的比较和分析，以提高算法设计与分析的能力。

本教材系按电子工业部的工科电子类专业教材 1991～1995 年编审出版规划，由计算机教材编审委员会征稿并推荐出版。责任编委为杨成忠。

本教材由东南大学朱静华担任主编，电子科技大学江庆林担任主审。

本课程的参考学时数为 72 学时，使用本教材时可根据需要，在讲述基本概念、逻辑特性、存储结构、基本运算的同时，穿插介绍相应的题例，而不必按教材中次序讲授。

本教材由朱静华编写前四章，扬州大学殷新春编写后七章，朱静华统编全稿。参加审阅工作的还有吴跃、叶见欣同志，他们都为本书提出许多宝贵意见，这里表示诚挚的感谢。由于作者水平有限，书中难免还存在一些缺点和错误，殷切希望广大读者批评指正。

编著者

1995 年 4 月

目　录

第一章 绪 论

1.1 内容提要

1.1.1 基本概念

定义 1.1 **数据**(data)是描述客观事物的数字、字符以及其它能输入到计算机中并可被计算机程序处理的符号的集合。

定义 1.2 **数据元素**(data element)是数据的基本单位,即数据集合中的个体。它是计算机的最小可存取单位,在计算机程序中通常作为一个整体进行考虑和处理。有时一个数据元素可由若干个**数据项**(data item)组成,数据项是数据的不可分割的最小单位,也是数据集合的最小可命名单位。

定义 1.3 **数据对象**(data object)是具有相同特性的数据元素的集合,是数据集合的一个子集。

定义 1.4 **数据结构**(data structure)是带有结构的元素的集合,它是一个二元组

$$\text{Data\_Structure}=(D,R)$$

其中:D 为数据元素的有限集合;R 为 D 上关系的有限集合。此处,R 集合中的关系指的是数据元素之间的逻辑关系,因此,又称数据的**逻辑结构**(logical structure)。

例如,复数可被定义为一种数据结构

$$\text{Complex}=(D,R)$$

其中:$D=\{x \mid x \text{ 是实数}\}$;$R=\{\langle x,y\rangle \mid x,y\in D, x \text{ 称为实部}, y \text{ 称为虚部}\}$。

同样,整数、实数也是数据结构。由于它们的数据元素之间的关系都很简单,可称它们为**初等数据结构**。

定义 1.5 数据的**物理结构**(physical structure),是数据的逻辑结构在计算机中的表示(或映象),它包含数据元素的映象和关系的映象。

通常把数据的逻辑结构简称为数据结构,数据的物理结构又称为**存储结构**(storage structure)。

定义 1.6 在计算机上处理信息的最小单位是一位二进制数,叫做**位**(bit)。在计算机中,可以用一个由若干位组合起来的位串表示一个数据元素,称这个位串为**元素**(element)或**结点**(node)。当数据元素由若干个数据项组成时,位串中对应于每个数据项的子位串称做**数据域**(data field,或称**数据字段**)。结点的域按其性质可分成两大类,一类是存放数据元素本身的域,一般称为**信息域**;另一类用来存放该结点与其它结点的连接信息,一般称为**指针域**。

定义 1.7 **数据类型**(data type)是程序设计语言中所允许的变量的种类。换句话说,是变量所能取的值和能进行的运算的集合。

可以把数据类型看作是程序设计语言中已实现的数据结构,因此,数据类型实际上是数据结构(包括逻辑结构和存储结构)及其运算的总称。

定义 1.8 **顺序存储结构**，就是将数据结构中各个数据元素按照某种顺序规则存放在计算机存储器的一组相继单元中。

顺序存储结构的明显特点是：结点只有信息域，而不需要表示连接信息的指针域，访问某个信息必须知道存储结构的地址（即第一个结点的地址）和该信息在存储结构中的相对位置。

顺序存储结构的优点是存储密度高（存储密度＝结点中信息域占据的存储空间/整个结点占据的存储空间），访问简便；缺点是插入、删除、合并等操作不方便。因此，顺序存储结构适合于大小固定不变，且更新较少的数据结构。

定义 1.9 **链式存储分配**，就是结点中至少包含一个指针域的地址分配方式。把这种分配方式所形成的存储结构叫做**链式结构**或**链表**(list)。

在链式存储分配中，数据结构的各数据元素不一定相继存储，但是就每个元素而言，它所占据的存储单元却是相继的，即元素本身采用顺序式分配。

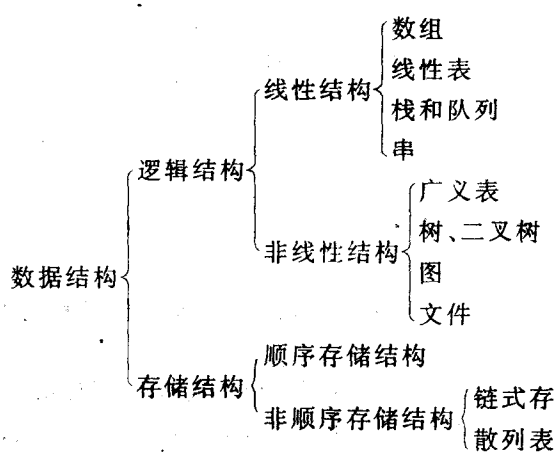
链式存储分配的优点是插入、删除、合并等操作比较方便；缺点是存储密度低，访问搜索较耗费时间。链式存储分配比较灵活，同一个元素可以处于两个甚至几个不同的链表中。

定义 1.10 **线性链表**，就是其结点结构中只含有一个指针域（用来指出其后继结点的位置）的结构。线性链表中有两个特殊指针：一是附加的头指针（first 或 head），该指针指向链表的第一个结点；另一个是“空”指针（**nil**，也记为 Λ），通常它是最后一个结点的指针域的值。

定义 1.11 **循环链表**(circular list)是指最后一个结点的指针域指向第一个结点的链表结构。也就是说最后一个结点的指针域不再是 **nil**，而是第一个结点的地址。

链式存储结构除线性链表、循环链表外，还有双向链表、双向链环等，另外在链表中还可以附加一个特殊的结点——头结点。

1.1.2 数据结构的分类



定义 1.12 **线性数据结构**指除头、尾外的每个数据元素都有一直接前趋和一直接后继。在**非线性数据结构**中，至少存在一个数据元素有不止一个前趋或后继。

每种逻辑结构都有与其相应的存储结构。同一种逻辑结构，在不同操作时，可以选用不同的存储结构。

1.1.3 算法的概念

定义 1.13 **算法**(algorithm)是指令的一个有限集合，它精确描述了解题方法。算法具有

如下性质：

(1) 有输入：一个算法有 0 个或 0 个以上的输入，即执行算法之前的初始值；

(2) 有输出：一个算法有 1 个或 1 个以上的输出，它们是与输入有某种特定关系的量；

(3) 有穷性：一个算法必须在执行有限步以后结束；

(4) 确定性：算法的每一步骤必须被确切定义，无二义性；

(5) 可行性：算法的每条指令都必须是基本的、可执行的，原则上可由人仅用笔和纸做有穷次运算或用计算机可实现的。

1.1.4 算法描述

描述算法的方法很多，本书主要使用伪形式语言——类 PASCAL 语言，其约定见附录。有时也用流程图或自然语言描述。

1.1.5 算法分析

除正确性外，衡量一个算法的好坏，通常主要考虑下列三个方面：

(1) 依据算法编制成程序后在计算机中运行时所耗费的时间；

(2) 依据算法编制成程序后在计算机中所占存储量的大小，其中主要考虑程序运行时所需辅助存储量的大小；

(3) 算法是否易读、是否容易转换成任何其它语言编制的可运行的程序以及是否易被测试等等。

一个程序在计算机上运行时所耗费的时间取决于下列因素：

(1) 程序运行时所需输入的数据总量；

(2) 对源程序进行编译所需时间；

(3) 计算机执行每条指令所需时间；

(4) 程序中的指令重复执行的次数。

前三条依赖于实现算法的计算机软、硬件系统，因此，习惯上常常把语句重复执行的次数作为算法的时间量度。

定义 1.14 所谓一个语句的**频度**(frequency count，或称**频数**)，即为该语句重复执行的次数。

定义 1.15 当问题的规模以某种单位由 1 增至 n 时，解决该问题的算法实现所占用的空间也以某种单位由 1 增到 $C_s f(n)$(其中 C_s 为常数，$f(n)$是 n 的函数)，以及运行算法所耗费的时间也以某种单位由 1 增至 $C_t g(n)$(其中 C_t 为常数，$g(n)$是 n 的函数)，则称 $O(f(n))$和 $O(g(n))$是该算法的**复杂性**(algorithm complexity)。前者称为**算法的空间复杂性**(space complexity)，后者称为算法的**时间复杂性**(time complexity)。

一般用程序中所有语句的频度之和的数量级作为时间复杂性。

然而，很多程序的运行时间不仅依赖于问题的规模，而且依赖于它所处理的数据集。对这类算法，除特别指明外，均依据各种可能出现的数据集中最坏的情况来估算算法的时间复杂性，有时也在某种约定(如等概率)下讨论算法的平均时间复杂性。

1.2 题 例

例 1.1 数据结构研究的主要问题是什么？

解 数据结构是研究数据元素之间抽象化的相互关系(逻辑结构)和这种关系在计算机中的存储表示(存储结构),并对这种结构定义相适应的运算,设计相应的算法,而且确保经过这些运算后所得到的新结构仍然是原来的结构。

例 1.2 填空。

(1) 数据结构的三要素是指__①__、__②__以及它们的__③__。

(2) 链式存储结构与顺序存储结构相比较,主要优点是__④__。

(3) 设有一批数据元素,为了最快地存取某元素,数据结构宜用__⑤__结构,为了方便地插入一个元素,数据结构宜用__⑥__结构。

解 ①数据元素;②逻辑结构;③映象(存储结构);④插入、删除、合并等操作较方便;⑤顺序存储;⑥链式存储。

例 1.3 举例说明:数据的逻辑结构与存储结构是相对而言的,在一定条件下,逻辑结构也可看成是存储结构。

解 一维数组可以被看作是一种逻辑结构,因为它反映了数据集合的两个特点:①具有有限个元素;②元素间具有顺序(前后)关系。如果用户采用汇编语言编程,所用的存储介质是存储器,则可以在一维数组和计算机存储器之间建立某种映射关系,比如,令它对应于存储器中的一组相继单元,把一维数组的元素按顺序存放于这组存储单元中,这一组相继的存储单元就是存放该数组的存储结构。但是,如果用户采用高级程序设计语言编程,则计算机存储器对用户来说是透明的,用户感觉到的只是逻辑存储器——简单变量和数组,用户可以用简单变量、数组来表达或存储信息。在这种情况下,数组既是逻辑结构,同时也被看作是存储结构。

例 1.4 考虑下面两段描述:

```
(1) procedure sam1;
    begin
      n:=2;
      while not Odd(n) do n:=n+2;
      Writeln(n)
    end;  {sam1}
```

```
(2) procedure sam2;
    begin
      y:=0;
      x:=5/y;
      Writeln(x,y)
    end;  {sam2}
```

这两段描述均不能满足算法的准则,试问它们违反了哪些准则?

解 (1) 违反了算法的有穷性要求,将导致过程不可终止。(2)违反了算法的可行性要求。因为5除以0在数学上得到的是无穷大,要将它赋给变量 x 是不可行的,机器将发生溢出错误。

例 1.5 下面三个算法片段中,假定列出的语句均不包含于任何显式或隐式的循环中,请写出语句 x:=x+1的频度。

```
(1) procedure a;
    begin
      ⋮
    x:=x+1;
      ⋮
    end;
```

```
(2) procedure b;
    begin
      ⋮
    for i:=1 to n do
      x:=x+1;
      ⋮
    end;
```

```
(3) procedure c;
    begin
      ⋮
    for i:=1 to n do
      for j:=1 to i do
        x:=x+1;
        ⋮
    end;
```

解 (1) 1; (2) $\sum_{i=1}^{n}1=n$; (3) $\sum_{i=1}^{n}\sum_{j=1}^{i}1=\sum_{i=1}^{n}i=(n^2+n)/2$。

例 1.6 将下列算法的时间复杂性级别，按照由低到高的顺序排成一行(n 是问题的规模)。

$O(n)$ $O(2^n)$ $O(\log_2 n)$ $O(n\log_2 n)$ $O(n^5)$ $O(n^2+1)$ $O(n^3-n^2)$ $O(1)$

解 $O(1)$表示计算时间为常数；$O(n)$为线性型；$O(n^2+1)$为平方型；$O(n^3-n^2)$为立方型；$O(2^n)$为指数型；$O(\log_2 n)$为对数型，在算法分析时，常将 $\log_2 n$ 写成 $\log n$。

当 n 足够大时，排列次序是：

$O(1)$ $O(\log n)$ $O(n)$ $O(n\log n)$ $O(n^2+1)$ $O(n^3-n^2)$ $O(n^5)$ $O(2^n)$

例 1.7 分析下列程序中各语句的频度。

```
procedure ex;
  begin
  ① for i:=1 to 9 do
  ②   Write (i);
  ③ for i:=1 to 2 do
  ④   Writeln;
  ⑤ for i:=1 to 9 do
  ⑥   [ for j:=1 to i do
  ⑦       [ p:=i*j;
  ⑧         Write(p)];
  ⑨     for j:=1 to 2 do
  ⑩         Writeln]
  end;
```

解 各语句频度分别是：①10；②$\sum_{i=1}^{9}1=9$；③3；④$\sum_{i=1}^{2}1=2$；⑤10；⑥$\sum_{i=1}^{9}(i+1)=54$；⑦、⑧$\sum_{i=1}^{9}\sum_{j=1}^{i}1=\sum_{i=1}^{9}i=45$；⑨$\sum_{i=1}^{9}3=27$；⑩$\sum_{i=1}^{9}\sum_{j=1}^{2}1=18$。

例 1.8 试确定下列程序段中各语句的频度。

```
①  i:=1;
②  while i≤n do
③    [ x:=x+1; i:=i+1]
```

解 各语句的频度分别为：①1；②n+1；③n。

例 1.9 设 n 为3的倍数，试分析以下程序段中第②、③、④语句的语句频度及程序段的时间复杂性。

```
①  for i:=1 to n do
②    if 3*i≤n then
③      for j:=3*i to n do
④        [x:=x+1; y:=3*x+2]
```

解 各语句的频度分别为：

②n；③$\sum_{i=1}^{n/3}(n-3i+2)=\frac{n(n+1)}{6}$；④$\sum_{i=1}^{n/3}(n-3i+1)=\frac{n(n-1)}{6}$；

程序段的时间复杂性为 $O(n^2)$。

例 1.10 试分析下列程序段中第①、②、③语句的语句频度及程序段的时间复杂性。

```
①  for i:=1 to n-1 do
```

```
②    for j:=n downto i+1 do
③      if a[j-1]>a[j] then
④        [t:=a[j-1]; a[j-1]:=a[j]; a[j]:=t];
```

解 各语句的频度分别为：

①n； ②$\sum_{i=1}^{n-1}(n-i+1)=\frac{(n-1)(n+2)}{2}$； ③$\sum_{i=1}^{n-1}(n-i)=\frac{n(n-1)}{2}$。

程序段的时间复杂性为$O(n^2)$。

例 1.11 试写一个算法，顺序读入3个整数 x、y 和 z，然后按自大至小的顺序输出。

解 假设仍依 x、y 和 z 的次序输出这3个整数，则本题的目标是使 x≥y≥z。在算法中应考虑对这三个元素作尽可能少的比较和移动。一个易读的算法如下：

```
procedure descending1;
  begin
    Read(x,y,z);
    if x<y then  {使 x≥y}
      [temp:=x; x:=y; y:=temp];
    if x<z then  {使 x≥z}
      [temp:=x; x:=z; z:=temp];  {此时 x 中的值最大}
    if y<z then  {使 y≥z}
      [temp:=y; y:=z; z:=temp];
    Write(x,y,z)
  end;  {descending1}
```

本算法在最坏情况下需进行3次比较和9次移动(赋值)。稍加改变，可得到在最坏情况下只需进行3次比较和7次移动的算法：

```
procedure descending2;
  begin
    Read(x,y,z);
    if x<y then  {使 x≥y}
      [temp:=x; x:=y; y:=temp];
    if y<z then
      [temp:=z; z:=y;  {此时 z 中的值最小}
        if x≥temp
          then y:=temp
          else [y:=x; x:=temp]];
    Write(x,y,z)
  end;  {descending2}
```

例 1.12 利用 PASCAL 语言中前趋函数 Pred 和后继函数 Succ 写出求 a+b 的递归函数。其中 a、b 为非负整数。

解 设计递归算法时要注意两点：一是递归结束条件，一是如何递归。显然，

$$a+b=\begin{cases}a & (\text{当 } b=0\text{时})\\ a+(b-1)+1 & (\text{当 } b\neq 0\text{时})\end{cases}$$

为加快速度，可用下列公式：

$$a+b=\begin{cases} b & (\text{当 } a=0\text{时}) \\ a & (\text{当 } b=0\text{时}) \\ a+(b-1)+1 & (\text{当 } a>b \text{ 时}) \\ (a-1)+b+1 & (\text{当 } a\leqslant b \text{ 时}) \end{cases}$$

其中，加1、减1运算分别通过 Succ 和 Pred 实现。所求算法为：

```
function aaddb(a,b:Integer): Integer;
  begin
    case
      a=0: aaddb:=b;
      b=0: aaddb:=a;
      else case compare(a,b) of
          '>'      : aaddb:=Succ(aaddb(a,Pred(b)));
          '<','=': aaddb:=Succ(aaddb(Pred(a),b))
          end
    end
  end; {aaddb}
```

其中，compare 函数后面要多次使用，请读者自行写出其定义。

例 1.13 写出一个递归过程，将读入的一列字符以逆顺序印出(以'.'结束)。

解

```
procedure revers;
  var c:Char;  {c 是本过程的局部量}
  begin
     Read(c);
     if c≠'.' then revers;
     Write(c)
  end; {revers}
```

例 1.14 若 n 为一正整数，试设计一个算法，确定 n 是否为它的所有因子之和。

解 算法一

```
procedure sad1(n:Integer);
    {验证 n 是否为其所有因子之和，n 为大于1的整数}
  begin
     t:=1; s:=0;  {s 中存放 n 的所有因子之和}
     while t<n do  {1≤t<n}
        [if n mod t=0 then s:=s+t;   {t 是 n 的因子}
          t:=t+1  {检查下一个 t}];
     if s=n
        then Write('n 是其所有因子之和')
        else Write('n 不是其所有因子之和')
  end;  {sad1}
```

算法二

```
procedure sad2(n:Integer);
  begin
     t:=2; s:=1;  {s 中存放 n 的所有因子之和}
     if Trunc(Sqrt(n))=Sqrt(n)  {此处简略表示 √n 是整数}
        then [i:=Sqrt(n); s:=s+i]
        else i:=Trunc(Sqrt(n));
```

```
        while t<i do  {最大的因子不超过 i}
            [ if n mod t=0 then
                  [r:=n div t; s:=s+r+t];  {r 和 t 均是 n 的因子}
              t:=t+1];
        if s=n
            then Write('n 是其所有因子之和')
            else Write('n 不是其所有因子之和')
    end;  {sad2}
```

算法 sad1直观易读，但执行时间长，循环语句要执行 n－1次，时间复杂性为 O(n)。算法 sad2的循环语句只要执行 $\sqrt{n}$ 次，时间复杂性为 O($\sqrt{n}$)。可见算法 sad2优于算法 sad1，当 n 较大时尤为明显。