新世纪

计算机基础教育丛书

丛书主编
谭浩强

计算机软件技术基础习题解答

徐士良 编著

清华大学出版社

新世纪
计算机基础教育丛书

丛书主编
谭浩强

计算机软件技术基础习题解答

徐士良 编著

清华大学出版社
北京

内 容 简 介

本书是《计算机软件技术基础》一书的辅助教材。书中给出了《计算机软件技术基础》一书中所有习题的参考解答，对有些习题还给出了详细分析。同时本书前3章在原有习题后面还适当增加了一些习题，便于读者学习有关内容。

图书在版编目(CIP)数据

计算机软件技术基础习题解答/徐士良编著. —北京：清华大学出版社，2003
(新世纪计算机基础教育丛书/谭浩强主编)
ISBN 7-302-07713-4

Ⅰ.计… Ⅱ.徐… Ⅲ.软件—高等学校—解题 Ⅳ.TP31-44

中国版本图书馆CIP数据核字(2003)第108657号

出 版 者：清华大学出版社　　地　　址：北京清华大学学研大厦
http://www.tup.com.cn　　邮　　编：100084
社 总 机：010-62770175　　客户服务：010-62776969
组稿编辑：焦　虹
文稿编辑：霍志国
印 刷 者：北京国马印刷厂
装 订 者：三河市金元装订厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　印张：9.5　字数：214千字
版　　次：2004年1月第1版　2004年1月第1次印刷
书　　号：ISBN 7-302-07713-4/TP·5648
印　　数：1～5000
定　　价：13.00元

序

Preface Preface Preface Preface

21世纪终于来临了，在新的世纪，人们自然对未来有许多美好的愿望和设想。现代科学技术的飞速发展，改变了世界，也改变了人类的生活。作为新世纪的大学生，应当站在时代发展的前列，掌握现代科学技术知识，调整自己的知识结构和能力结构，以适应社会发展的要求。新世纪需要具有丰富的现代科学知识、能够独立解决面临的任务、充满活力、有创新意识的新型人才。

掌握计算机知识和应用无疑是培养新型人才的一个重要环节。计算机既是现代科学技术的结晶，又是大众化的工具。学习计算机知识不仅是为了掌握一种技能，更重要的是：它能启发人们对先进科技的向往，激发创新意识，推动对新知识的学习，培养自学能力，锻炼动手实践的本领。因而它是高等学校全面素质教育中极为重要的一部分。

自20世纪80年代初以来，高等学校中计算机教育（尤其是非计算机专业中的计算机教育）发展迅速，计算机教育的内容不断扩展，程度不断提高，它所起的作用也愈来愈显著。

在实践中，大家已认识到，计算机应用人才队伍是由两部分人组成的：一部分是计算机专业出身的计算机专业人才，他们是计算机应用人才队伍中的骨干力量；另一部分是各行各业中应用计算机的人员。这后一部分人一般并非从计算机专业毕业，他们人数众多，既熟悉自己所从事的专业，又掌握计算机的应用知识，善于用计算机作为工具去解决本领域中的任务。他们是计算机应用人才队伍中的基本力量。事实上，大部分应用软件都是由非计算机专业出身的计算机应用人员研制的。他们具有的这个优势是其他人难以代替的。从这个事实可以看到在非计算机专业中深入进行计算机教育的必要性。

非计算机专业中的计算机教育，无论目的、内容、教学体系、教材、教学方法等各方面都与计算机专业有很大的不同，决不应该照搬计算机专业的模式和做法。全国高等院校计算机基础教育研究会自1984年成立以来，始终不渝地探索高校计算机基础教育的特点和规律，在20世纪80年代中期，最早提出了按层次进行教育的方案。计算机应用是分层次的，不同的人在不同的层次上使用着计算机；同样，计算机教育也是分层次的，以适应不同应用层次的要求。全国有一千多所高等学校，好几百个专

业，学校的类型、条件和基础差别很大，不可能按同一模式、同一要求、同一内容进行教学。按层次组织教学，可以使不同专业、不同学校根据自己的情况选择教学内容，做到“各取所需”。

经过十多年的实践，几经调整，许多高校形成了按以下三个层次组织教学的方案：第一层次为计算机公共基础，学习计算机基本知识和基本操作；第二层次为计算机技术基础，内容包括程序设计、数据库、网络和多媒体技术等；第三层次为计算机应用基础，结合专业应用的需要学习有关计算机应用课程。每一层次中设立若干门课程，包括必修课和选修课。

1988年起，我们根据层次教学的方案，组织编写了“计算机基础教育丛书”，邀请有丰富教学经验的专家学者先后编写了20多种教材，由清华大学出版社出版。丛书出版后，迅速受到广大高校师生的欢迎，对高等学校的计算机基础教育起了积极的推动作用。广大读者反映这套教材定位准确、内容丰富、通俗易懂，符合广大非计算机专业学生的特点。许多高校都采用了我们编写的教材。丛书总发行量达到700多万册，这在全国是罕见的。

在新世纪来临之际，我们在该丛书成功的基础上组织编写了这套“新世纪计算机基础教育丛书”，以适应新形势的要求。本丛书有以下特点：

（1）内容新颖。根据新世纪的需要，重新确定丛书的内容，以符合计算机科学技术的发展和教学改革的要求。本丛书除保留了原丛书中经过实践考验，且深受群众欢迎的优秀教材外，还编写了许多新的教材，在这些教材中反映了近年来迅速得到推广应用的一些计算机新技术，以后还将根据发展不断补充新的内容。

（2）适合按层次组织教学的需要。在新世纪大多数学校是采用层次教学模式的，但不同的学校和专业所达到的层次不同。本丛书采用模块形式，提供了各种课程的教材，内容覆盖高校计算机基础教育的三个层次。丛书中既有供理工类专业用的教材，也有供文科和经济类专业用的教材；既有必修课的教材，也包括一些选修课的教材。各类学校都可以从中选择到合适的教材。

（3）符合大学非计算机专业学生的特点。本丛书针对非计算机专业学生的特点，以应用为目的，以应用为出发点，强调实用性。本丛书的作者都是长期在第一线从事高校计算机基础教育的教授和副教授，对学生的基础、特点和认识规律有深入的研究，在教学实践中积累了丰富的经验，可以说，每一本教材都是他们长期教学经验的总结。在教材的写法上，既注意概念的严谨和清晰，又特别注意采用读者容易理解的方法阐明看似深奥难懂的问题，做到例题丰富、通俗易懂、便于自学。这一点是本丛书一个十分重要的特点。书是写给读者看的，读者如果看不懂，只能算

写作的失败。

(4) 采用多样化的形式。除了文字教材这一基本形式外，有些教材还配有习题解答和上机指导。我们还准备采用现代教学方式，陆续制作电子出版物，以利于学生自学。

总之，本丛书的指导思想是：内容新颖、概念清晰、实用性强、通俗易懂、层次配套。简单概括为："新颖、清晰、实用、通俗、配套"。我们经过多年实践形成的这一套行之有效的创作风格相信会受到广大读者的欢迎。判别一本书的优劣，读者最有发言权。

本丛书多年来得到了各方面人士的指导、支持和帮助，尤其是得到了全国高等院校计算机基础教育研究会的各位专家和各高校老师们的支持和帮助，我们在此表示由衷的感谢。

本丛书肯定会有不足之处，竭诚希望得到广大读者的批评指正。

丛 书 主 编

全国高等院校计算机基础教育研究会理事长

谭 浩 强

2000 年 1 月 1 日

前言

Foreword Foreword Foreword Foreword

《计算机软件技术基础》一书出版以后，受到了广大读者的欢迎，许多读者希望给出该书中所有习题的参考答案。为了满足读者的要求，作者编写了本书，将《计算机软件技术基础》一书中的所有习题按章给出参考答案，对有些习题还给出了详细分析。为进一步帮助读者理解基本概念，除了给出书中原有习题的答案外，在前3章中还增加了一些习题，把它们放在原习题的后面，同时也给出了解答。

对于要求编写算法的习题，除了采用《计算机软件技术基础》一书中所介绍的算法描述语言描述外，同时还给出了C语言描述。书中的大部分算法都用C语言编程调试通过。由于时间紧迫与水平有限，书中难免有错误或不妥之处，恳请读者批评指正。

作　者

2003年11月

目 录

Catalog Catalog Catalog Catalog

第1章 算 法

1.1 设给定3个整数a,b,c,试写出寻找3个整数的中数的算法；并分析在平均情况与最坏情况下,该算法分别要做多少次比较?

解：寻找给定3个整数的中数的算法如下所述。

输入：给定的3个整数a,b,c。

输出：给定3个整数中的中数m。

```
PROCEDURE  MID(a,b,c,m)
m=a;
IF  m≥b  THEN
  { IF  m≥c  THEN
    { IF  b≥c  THEN  m=b        [b为中数]
      ELSE  m=c                 [c为中数]
    }
  }
ELSE
  { IF  m≤c  THEN
    { IF  b≥c  THEN  m=c        [c为中数]
      ELSE  m=b                 [b为中数]
    }
  }
RETURN
```

上述算法用C语言描述如下(中数m由函数值返回)：

```
int  mid(int a,int b,int c)
{ int m;
  m=a;
  if (m>=b)
    { if (m>=c)
        { if (b>=c)  m=b;       /* b为中数*/
          else  m=c;            /* c为中数*/
        }
    }
  else
    { if (m<=c)
        { if (b>=c)  m=c;       /* c为中数*/
          else  m=b;            /* b为中数*/
        }
```

```
    }
  return(m);                    /* 返回中数 m */
}
```

假设 a,b,c 中的每一个数为中数的概率相等(均为 1/3)。由于当 a 为中数时需要比较 2 次,b 或 c 为中数时均需要比较 3 次,因此,在平均情况下上述算法所需要的比较次数为

$$2\times(1/3)+3\times(1/3)+3\times(1/3)=8/3$$

即在平均情况下,上述算法需要比较 8/3 次。

在最坏情况下,上述算法需要比较 3 次(当 b 或 c 为中数时)。

1.2 利用减半递推技术,写出求长度为 n 的数组中最大元素的递归算法。设 $n=2^k$,其中 $k\geqslant 1$。

解:利用减半递推技术,写出求数组 A(m: n)中最大元素的递归过程。

如果数组中只有一个元素,则该元素即是数组中最大的元素,否则将数组对分为前半部分和后半部分:

(1) 用同样的方法求数组前半部分的最大值 max1。

(2) 用同样的方法求数组后半部分的最大值 max2。

(3) 若 max1>max2,则 max1 为数组中的最大值;否则 max2 为数组中的最大值。

上述过程用算法语言描述如下所述。

输入:数组 A(m: n)。

输出:数组 A 中的最大元素值 max。

```
PROCEDURE  MAXA(A,m,n,max)
IF (m=n)  THEN  max=A(m)
ELSE
  { MAXA(A,m,(m+n)/2,max1)        [求数组前半部分的最大值 max1]
    MAXA(A,(m+n)/2+1,n,max2)      [求数组后半部分的最大值 max2]
    IF (max1>max2) THEN  max=max1
    ELSE max=max2
  }
RETURN
```

上述算法用 C 语言描述如下(包括主函数):

```
maxa(a,m,n)
int m,n,a[];
{ int d,d1,d2;
  if (m==n) return(a[m-1]);
  else
    { d1=maxa(a,m,(m+n)/2);          /*求数组前半部分的最大值 d1*/
      d2=maxa(a,(m+n)/2+1,n);        /*求数组后半部分的最大值 d2*/
      if (d1>d2) d=d1;
      else d=d2;
```

```
        return(d);
      }
  }

  #include "stdio.h"
  main()
  { int a[16]={15,6,4,-1,5,21,7,3,78,-51,40,36,43,49,63,27};
    printf("max=%d\n",maxa(a,1,16));
  }
```

上述C程序的运行结果为

max=78

补充习题：

1.3 设有12个小球。其中11个小球的重量相同，称为好球；有一个小球的重量与11个好球的重量不同(或轻或重)，称这个小球为坏球。试编写一个算法，用一个无砝码的天平称3次找出这个坏球，并确定其比好球轻还是重。

解：用一个长度为12的整型数组A(1:12)模拟表示编号分别为1～12的12个小球，其中的元素值分别是各小球的重量。即在这个整型数组中，11个元素值是相同的，称为好元素；只有一个元素的值与11个好元素值不同(其值或大或小)，称为坏元素。

下面通过对数组中元素值的比较来找出这个坏元素。

首先将12个元素分成以下3组：第1组为A(1),A(2),A(3),A(4)4个元素；第2组为A(5),A(6),A(7),A(8)4个元素；第3组为A(9),A(10),A(11),A(12)4个元素。

3次比较过程如表1.1所示。

表1.1 比较过程

第1次比较	第2次比较	第3次比较
若A(1)+A(2)+A(3)+A(4)=A(5)+A(6)+A(7)+A(8) 则A(9),A(10),A(11),A(12)中有坏	若A(1)+A(9)=A(10)+A(11) 则A(12)坏	若A(1)>A(12),则A(12)坏且轻
		若A(1)<A(12),则A(12)坏且重
	若A(1)+A(9)>A(10)+A(11) 则A(9)坏且重 或A(10)与A(11)有坏且轻	若A(10)=A(11),则A(9)坏且重
		若A(10)>A(11),则A(11)坏且轻
		若A(10)<A(11),则A(10)坏且轻
	若A(1)+A(9)<A(10)+A(11) 则A(9)坏且轻 或A(10)与A(11)有坏且重	若A(10)=A(11),则A(9)坏且轻
		若A(10)>A(11),则A(10)坏且重
		若A(10)<A(11),则A(11)坏且重

续表

第 1 次比较	第 2 次比较	第 3 次比较
若 A(1)+A(2)+A(3)+A(4)>A(5)+A(6)+A(7)+A(8) 则 A(1),A(2),A(3),A(4)中有坏,且为重 或 A(5),A(6),A(7),A(8)中有坏,且为轻	若 A(1)+A(2)+A(6)=A(5)+A(3)+A(4) 则 A(7)与 A(8)中有坏且轻	若 A(1)=A(7),则 A(8)坏且轻
		若 A(1)≠A(7),则 A(7)坏且轻
	若 A(1)+A(2)+A(6)>A(5)+A(3)+A(4) 则 A(1),A(2)中有坏且重 或 A(5)坏且轻	若 A(1)=A(2),则 A(5)坏且轻
		若 A(1)>A(2),则 A(1)坏且重
		若 A(1)<A(2),则 A(2)坏且重
	若 A(1)+A(2)+A(6)<A(5)+A(3)+A(4) 则 A(3),A(4)中有坏且重 或 A(6)坏且轻	若 A(3)=A(4),则 A(6)坏且轻
		若 A(3)>A(4),则 A(3)坏且重
		若 A(3)<A(4),则 A(4)坏且重
若 A(1)+A(2)+A(3)+A(4)<A(5)+A(6)+A(7)+A(8) 则 A(1),A(2),A(3),A(4)中有坏,且为轻 或 A(5),A(6),A(7),A(8)中有坏,且为重	若 A(1)+A(2)+A(6)=A(5)+A(3)+A(4) 则 A(7)与 A(8)中有坏且重	若 A(1)=A(7),则 A(8)坏且重
		若 A(1)≠A(7),则 A(7)坏且重
	若 A(1)+A(2)+A(6)>A(5)+A(3)+A(4) 则 A(3),A(4)中有坏且轻 或 A(6)坏且重	若 A(3)=A(4),则 A(6)坏且重
		若 A(3)>A(4),则 A(4)坏且轻
		若 A(3)<A(4),则 A(3)坏且轻
	若 A(1)+A(2)+A(6)<A(5)+A(3)+A(4) 则 A(1),A(2)中有坏且轻 或 A(5)坏且重	若 A(1)=A(2),则 A(5)坏且重
		若 A(1)>A(2),则 A(2)坏且轻
		若 A(1)<A(2),则 A(1)坏且轻

根据表 1.1 所示的比较过程,其算法如下所述。

输入:整型数组 A(1:12)。

输出:坏元素的下标 k。当 k>0 时,表示该坏元素比好元素重;当 k<0 时,表示该坏元素比好元素轻。

```
PROCEDURE (A,k)
IF (A(1)+A(2)+A(3)+A(4)=A(5)+A(6)+A(7)+A(8)) THEN
                          [A(9),A(10),A(11),A(12)中有坏]
    IF (A(1)+A(9)=A(10)+A(11))        THEN [A(12)坏]
        IF (A(1)>A(12)) THEN { k=-12; RETURN } [A(12)坏且轻]
        ELSE { k=12; RETURN } [A(12)坏且重]
    ELSE IF (A(1)+A(9)>A(10)+A(11)) THEN
                          [A(9)坏且重,或 A(10)与 A(11)中有坏且轻]
        IF (A(10)=A(11)) THEN { k=9; RETURN } [A(9)坏且重]
        ELSE IF (A(10)>A(11)) THEN { k=-11; RETURN }[A(11)坏且轻]
```

```
            ELSE { k=-10; RETURN } [A(10)坏且轻]
        ELSE    [A(9)坏且轻,或 A(10)与 A(11)中有坏且重]
            IF (A(10)=A(11))  THEN  { k=-9; RETURN }  [A(9)坏且轻]
            ELSE IF (A(10)>A(11)) THEN { k=10; RETURN } [A(10)坏且重]
            ELSE { k=11; RETURN } [A(11)坏且重]
    ELSE IF (A(1)+A(2)+A(3)+A(4)>A(5)+A(6)+A(7)+A(8))  THEN
        [A(1),A(2),A(3),A(4)中有坏且为重; 或 A(5),A(6),A(7),A(8)中有坏且为轻]
        IF (A(1)+A(2)+A(6)=A(5)+A(3)+A(4))  THEN [A(7),A(8)中有坏且为轻]
            IF (A(1)=A(7))  THEN  { k=8; RETURN }        [A(8)坏且为轻]
            ELSE  { k=7; RETURN }                        [A(7)坏且为轻]
        ELSE IF (A(1)+A(2)+A(6)>A(5)+A(3)+A(4))  THEN
                                    [A(1),A(2)中有坏且为重; 或 A(5)坏且为轻]
            IF (A(1)==A(2))  THEN  { k=5; RETURN }       [A(5)坏且为轻]
            ELSE IF (A(1)>A(2))  THEN  { k=1; RETURN }   [A(1)坏且为重]
            ELSE { k=2; RETURN }                         [A(2)坏且为重]
        ELSE    [A(3),A(4)中有坏且为重; 或 A(6)坏且为轻]
            IF (A(3)==A(4))  THEN  { k=6; RETURN }       [A(5)坏且为轻]
            ELSE IF (A(3)>A(4))  THEN  { k=3; RETURN }   [A(3)坏且为重]
            ELSE  { k=4; RETURN }                        [A(4)坏且为重]
    ELSE [A(1),A(2),A(3),A(4)中有坏且为轻; 或 A(5),A(6),A(7),A(8)中有坏且为重]
        IF (A(1)+A(2)+A(6)=A(5)+A(3)+A(4))  THEN  [A(7),A(8)中有坏且为重]
            IF (A(1)=A(7))  THEN  { k=8; RETURN }        [A(8)坏且为重]
            ELSE  { k=7; RETURN }  [A(7)坏且为重]
        ELSE IF (A(1)+A(2)+A(6)>A(5)+A(3)+A(4))  THEN
                                    [A(3),A(4)中有坏且为轻; 或 A(6)坏且为重]
            IF (A(3)=A(4))  THEN  { k=6; RETURN }  [A(6)坏且为重]
            ELSE IF (A(3)>A(4))  THEN  { k=4; RETURN }  [A(4)坏且为轻]
            ELSE { k=3; RETURN }  [A(3)坏且为轻]
        ELSE        [A(1),A(2)中有坏且为轻; 或 A(5)坏且为重]
            IF (A(1)=A(2))  THEN  { k=5; RETURN }  [A(5)坏且为重]
            ELSE IF (A(1)>A(2))  THEN  { k=2; RETURN }  [A(2)坏且为轻]
            ELSE  { k=1; RETURN }  [A(1)坏且为轻]
}
```

上述算法用 C 语言描述如下(包括提供测试用例的主函数):

```
int a12(a)
int a[];
{
  if (a[1]+a[2]+a[3]+a[4]==a[5]+a[6]+a[7]+a[8])
                          /* A(9),A(10),A(11),A(12)中有坏 */
    if (a[1]+a[9]==a[10]+a[11])  /* A(12)坏 */
      if (a[1]>a[12])  return(-12); /* A(12)坏且轻 */
      else  return(12); /* A(12)坏且重 */
```

```
    else if (a[1]+a[9]>a[10]+a[11])
                        /* A(9)坏且重,或 A(10)与 A(11)中有坏且轻 */
        if (a[10]==a[11])  return(9); /* A(9)坏且重 */
        else if (a[10]>a[11])  return(-11); /* A(11)坏且轻 */
        else  return(-10); /* A(10)坏且轻 */
    else  /* A(9)坏且轻,或 A(10)与 A(11)中有坏且重 */
        if (a[10]==a[11])  return(-9); /* A(9)坏且轻 */
        else if (a[10]>a[11])  return(10); /* A(10)坏且重 */
        else  return(11); /* A(11)坏且重 */
else if (a[1]+a[2]+a[3]+a[4]>a[5]+a[6]+a[7]+a[8])
    /* A(1),A(2),A(3),A(4)中有坏且为重;或 A(5),A(6),A(7),A(8)中有坏且为轻 */
      if (a[1]+a[2]+a[6]==a[5]+a[3]+a[4])  /* A(7),A(8)中有坏且为轻 */
          if (a[1]==a[7])  return(-8);   /* A(8)坏且为轻 */
          else  return(-7);    /* A(7)坏且为轻 */
      else if (a[1]+a[2]+a[6]>a[5]+a[3]+a[4])
                          /* A(1),A(2)中有坏且为重;或 A(5)坏且为轻 */
          if (a[1]==a[2])  return(-5);   /* A(5)坏且为轻 */
          else if (a[1]>a[2])  return(1);   /* A(1)坏且为重 */
          else return(2);          /* A(2)坏且为重 */
      else   /* A(3),A(4)中有坏且为重;或 A(6)坏且为轻 */
          if (a[3]==a[4])  return(-6);   /* A(6)坏且为轻 */
          else if (a[3]>a[4])  return(3);   /* A(3)坏且为重 */
          else  return(4);          /* A(4)坏且为重 */
else
  /* A(1),A(2),A(3),A(4)中有坏且为轻;或 A(5),A(6),A(7),A(8)中有坏且为重 */
      if (a[1]+a[2]+a[6]==a[5]+a[3]+a[4]) /* A(7),A(8)中有坏且为重 */
          if (a[1]==a[7])  return(8);   /* A(8)坏且为重 */
          else  return(7);   /* A(7)坏且为重 */
      else if (a[1]+a[2]+a[6]>a[5]+a[3]+a[4])
                          /* A(3),A(4)中有坏且为轻;或 A(6)坏且为重 */
          if (a[3]==a[4])  return(6);   /* A(6)坏且为重 */
          else if (a[3]>a[4])  return(-4);   /* A(4)坏且为轻 */
          else return(-3);   /* A(3)坏且为轻 */
      else   /* A(1),A(2)中有坏且为轻;或 A(5)坏且为重 */
          if (a[1]==a[2])  return(5);   /* A(5)坏且为重 */
          else if (a[1]>a[2])  return(-2);   /* A(2)坏且为轻 */
          else  return(1);   /* A(-1)坏且为轻 */
}

#include "stdio.h"
main()
{ int k;
  static int a[13]={0,5,5,5,5,5,5,5,4,5,5,5,5};
```

```
    k=al2(a);
    if (k>)  printf("a[%d]=%d\n",k,a[k]);
    else  printf("a[%d]=%d\n",k,a[abs(k)]);
}
```

在上述C程序中，为了直观起见，定义的数组长度为13，其中数组元素a[0]在程序中不用。

上述程序运行的结果为

a[-8]=4 （表示数组中第8个元素为坏元素，即编号为8的小球为坏球，且比好球轻）

第 2 章　基本数据结构及其运算

2.1　什么叫数据结构？数据结构对算法有什么影响？请举例说明。

解：数据结构是指相互有关联的数据元素的集合。因此，一个数据结构既要反映数据元素的信息，还要反映数据元素之间的关系。数据元素之间的关系可以是逻辑关系（通常用前后件关系来表示），也可以是数据元素在计算机中的存储位置。

反映数据元素之间逻辑关系的数据结构称为数据的逻辑结构。数据的逻辑结构通常表示为 B=(D,R)，其中 B 为数据结构，D 为数据元素的集合，R 为反映 D 中各数据元素之间前后件关系的二元组的集合。

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，又称为数据的物理结构。

同一批数据元素的集合，采用不同的数据结构（特别是存储结构），其数据处理的效率是不一样的，主要体现在算法的时间复杂度与空间复杂度方面。下面举例来说明数据结构对算法效率的影响。

设数组 A 为非负整型数组，现要做如下两个运算：

（1）给定数组元素下标 i 的值，求相应的数组元素 A(i)的值。

（2）给定数组中某元素的值 X，求该数组元素的下标 i 的值（若值为 X 的数组元素有多个，则可任取一个）。

如果在计算机中已经顺序存储了数组 A 中的各元素值，则对于问题(1)来说，只需要做一个简单的运算即可，其算法如下：

算法 1　求数组元素 A(i)的值。

输入：顺序存储的数组 A，下标值 i。

输出：A(i)的值。

```
PROCEDURE  S1
INPUT  i
OUTPUT  A(i)
RETURN
```

但对于问题(2)来说，有可能需要搜寻整个数组 A，其算法如下：

算法 2　求值为 X 的数组元素的下标 i。

输入：顺序存储的数组 A，数组元素值 X。

输出：数组元素值为 X 的下标值 i。

```
PROCEDURE  S2
INPUT  X
i=1
```

```
WHILE  A(i)≠X  DO  i=i+1
OUTPUT  i
RETURN
```

比较上述算法1与算法2，明显可以看出，解决问题(2)比解决问题(1)要费时。但是，解决问题的实际难易程度还与数组A在计算机中的表示方法(即存储形式)有着密切的关系。在上述的算法中，求解问题(2)比求解问题(1)费时，是因为数组A中的各元素是以线性阵列来表示的，即数组A在计算机中的表示是以下标为顺序依次存放其中的各元素。如果以另外的方法来表示该数组，情况就会有所不同。例如，设数组A中的各元素值介于0与N之间，则可以用一个整型数组VALUE(0：N)来表示数组A，且满足如下条件：

若VALUE(X)＝i，则表示A(i)＝X；

若VALUE(X)＝－1，则表示无相应的i使A(i)＝X。

这样，对于问题(2)的求解也只需要做简单运算就可以了，其算法如下：

算法3　求值为X的数组元素的下标i。

输入：存储数组A信息的整型数组VALUE(0：N)，数组元素值X。

输出：数组A中元素值为X的下标值。

```
PROCEDURE  S3
INPUT  X
OUTPUT  VALUE(X)
RETURN
```

当输出值为－1时，说明值为X的数组元素不存在。当然，在这种表示方法中，数组A中的某些信息会丢失，因为在数组A中可能有多个元素具有相同的值。

由上述例子可以看出，一个算法的效率一般还与数据在计算机中的表示方法有直接的关系。

2.2　试写出在顺序存储结构下逆转线性表的算法，要求使用最少的附加空间。

解：顺序存储结构下逆转线性表的算法如下所述。

输入：长度为n的线性表数组A(1：n)。

输出：逆转后的长度为n的线性表数组A(1：n)。

```
PROCEDURE  INVSL(n,A)
FOR  k=1  TO  n/2  DO
  { t=A(k); A(k)=A(n-k+1); A(n-k+1)=t; }
RETURN
```

这个算法是原地工作的。

上述算法用C语言描述如下(其中ET为数据元素的类型)：

```
invsl(n,a)
int  n;
ET  a[];
{ int  k;
  ET  t;
```