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（第三版）习题解答

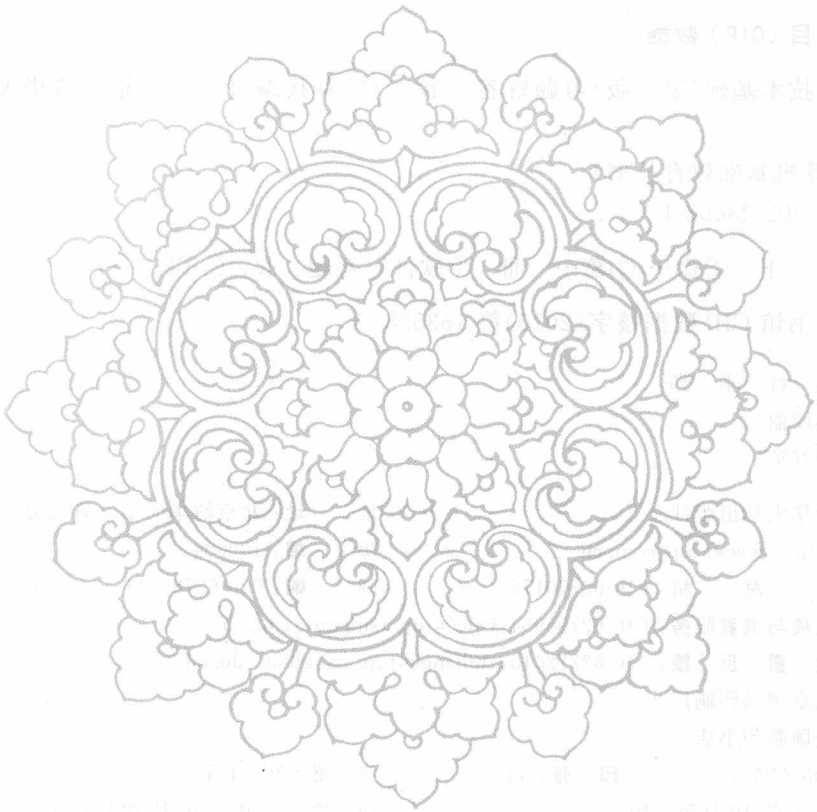
徐士良 葛兵 编著

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（第三版）习题解答

徐士良 葛兵 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》(第三版)一书的辅助教材。本书给出了《计算机软件技术基础》(第三版)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。

图书在版编目(CIP)数据

计算机软件技术基础(第三版)习题解答 / 徐士良,葛兵编著. —北京:清华大学出版社,2010.10
(新世纪计算机基础教育丛书)
ISBN 978-7-302-23408-1

Ⅰ. ①计… Ⅱ. ①徐… ②葛… Ⅲ. ①软件—解题 Ⅳ. ①TP31-44

中国版本图书馆 CIP 数据核字(2010)第 153598 号

责任编辑:焦 虹 李 晔
责任校对:焦丽丽
责任印制:李红英

出版发行:清华大学出版社 **地 址**:北京清华大学学研大厦 A 座
http://www.tup.com.cn **邮 编**:100084
社 总 机:010-62770175 **邮 购**:010-62786544
投稿与读者服务:010-62795954,jsjjc@tup.tsinghua.edu.cn
质 量 反 馈:010-62772015,zhiliang@tup.tsinghua.edu.cn
印 装 者:北京国马印刷厂
经 销:全国新华书店
开 本:185×260 **印 张**:11 **字 数**:262 千字
版 次:2010 年 10 月第 1 版 **印 次**:2010 年 10 月第 1 次印刷
印 数:1~3000
定 价:18.00 元

产品编号:039327-01

丛书序言

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，能够独立完成面临的任务，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。现在计算机技术已深入到人类生活的各个角落，与其他学科紧密结合，成为推动各学科飞速发展的有力的催化剂。无论学什么专业的学生，都必须具备计算机的基础知识和应用能力。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，不仅能够掌握有关知识，而且能培养人们的信息素养。这是高等学校全面素质教育中极为重要的一部分。

高校计算机基础教育应当遵循的理念是：面向应用需要；采用多种模式；启发自主学习；重视实践训练；加强创新意识；树立团队精神，培养信息素养。

计算机应用人才队伍由两部分人组成：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，绝不能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。2004 年，全国高等院校计算机基础教育研究会与清华大学出版社共同推出了《中国高等院校计算机基础教育课程体系 2004》（简称 CFC2004）；2006 年、2008 年又共同推出了《中国高等院校计算机基础教育课程体系 2006》（简称 CFC2006）及《中国高等院校计算机基础教育课程体系 2008》（简称 CFC2008），由清华大学出版社正式出版发行。

1988年起，我们根据教学实际的需要，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，符合大学生的特点。

1999年，根据新世纪的需要，在原有基础上组织出版了《新世纪计算机基础教育丛书》。由于内容符合需要，质量较高，被许多高校选为教材。丛书总发行量1000多万册，这在国内是罕见的。

最近，我们又对丛书作了进一步的修订，根据发展的需要，增加了新的书目和内容。本丛书有以下特点：

(1) 内容新颖。根据21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) 适合不同学校组织教学的需要。本丛书采用模块形式，提供了各种课程的教材，内容覆盖了高校计算机基础教育的各个方面。丛书中既有理工类专业的教材，也有文科和经济类专业的教材；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) 符合初学者的特点。本丛书针对初学者的特点，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师，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。

(4) 采用多样化的形式。除了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，并提供电子教案。

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、教材配套。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。

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。

本丛书肯定有不足之处，希望得到广大读者的批评指正。

欢迎访问谭浩强网站：http://www.tanhaoqiang.com

丛书主编

全国高等院校计算机基础教育研究会会长

谭浩强

前言

《计算机软件技术基础》一书出版以后，得到了广大读者的欢迎，并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，作者在修订《计算机软件技术基础》一书时，同时将《计算机软件技术基础》(第三版)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。

对于要求编写算法的习题，根据题目的特点，有的将同时给出面向过程和面向对象的 C++ 描述。

书中的大部分算法都用 C++ 语言编程调试过。

由于时间紧迫与水平有限，书中难免有错误或不妥之处，肯请读者批评指正。

作　者

2010 年 6 月

目 录

第1章 预备知识

1.1 设集合 $M=\{d_1,d_2,d_3,d_4,d_5,d_6\}$ 上的一个关系 R 如下：

$$R=\{(d_1,d_2),(d_2,d_4),(d_5,d_4),(d_1,d_5),(d_1,d_4),(d_1,d_6),(d_1,d_3),(d_3,d_6)\}$$

试验证

$$T_1=\{(d_1,d_2),(d_2,d_4),(d_1,d_3),(d_3,d_6),(d_5,d_4),(d_1,d_5)\}$$

是 R 的具有 6 个元素的基，而

$$T_2=\{(d_2,d_4),(d_1,d_3),(d_1,d_2),(d_6,d_5)\}$$

不是 R 的基。

解：对于 T_1，显然满足 $T_1\subseteq R$，且对于关系 R 中的每一个二元组 (x,y)，在 M 中存在元素 $x_0,x_1,x_2,\cdots,x_n$，满足主教材中定义 1.6 中的三个条件。验证过程如下：

R	$x_0,x_1,x_2,\cdots,x_n$	
(d_1,d_2)	d_1,d_2	$(d_1,d_2)\in T_1$
(d_2,d_4)	d_2,d_4	$(d_2,d_4)\in T_1$
(d_5,d_4)	d_5,d_4	$(d_5,d_4)\in T_1$
(d_1,d_5)	d_1,d_5	$(d_1,d_5)\in T_1$
(d_1,d_4)	d_1,d_2,d_4	$(d_1,d_2),(d_2,d_4)\in T_1$
(d_1,d_6)	d_1,d_3,d_6	$(d_1,d_3),(d_3,d_6)\in T_1$
(d_1,d_3)	d_1,d_3	$(d_1,d_3)\in T_1$
(d_3,d_6)	d_3,d_6	$(d_3,d_6)\in T_1$

由上可知，T_1 是 R 的具有 6 个元素的基。

对于 T_2，由于 $(d_6,d_5)\notin R$，因此，T_2 不是 R 的基。

1.2 设给定 3 个整数 a,b,c，试写出寻找这 3 个整数的中数的一个算法(用 C/C++ 描述)。并分析在平均情况与最坏情况下，你的算法分别要作多少次比较？

解：算法的 C++ 描述如下：

```
//ch1_1.cpp
#include<iostream>
using namespace std;
int mid(int a,int b,int c)
{ int m;
  m=a;                          //预设 a 为中数
  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
}
```

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)。

主函数例如下:

```
//主函数
int main()
{ int a,b,c;
  cout<<"input a,b,c=";
  cin>>a>>b>>c;
  cout<<"m="<<mid(a,b,c)<<endl;
  return 0;
}
```

运行结果如下(带有下划线的为键盘输入):

```
input a,b,c=4 8 3
m=4
```

1.3 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(用C/C++描述)。设$n=2^k$,其中$k\geqslant 1$。

解:利用减半递推技术,求数组中最大元素的递归过程如下:

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:

用同样的方法求数组前半部分的最大值max1。

用同样的方法求数组后半部分的最大值max2。

若max1>max2,则max1为数组中的最大值;

否则max2为数组中的最大值。

算法的C++描述如下:

```
//ch1_2.cpp
#include<iostream>
```

```
using namespace std;
int maxa(int a[],int m,int n)
{ int d,d1,d2;
  if (m==n) return(a[m-1]);
  else
  { d1=maxa(a,m,(m+n)/2);          //求数组中前半部分的最大值
    d2=maxa(a,(m+n)/2+1,n);        //求数组中后半部分的最大值
    if (d1>d2) d=d1;
    else d=d2;
    return(d);
  }
}
```

主函数例如下：

```
//主函数
int main()
{ int a[16]={15,6,4,-1,5,21,7,3,78,-51,40,36,43,49,63,27};
  cout<<"max="<<maxa(a,1,16)<<endl;
  return 0;
}
```

运行结果如下：

```
max=78
```

1.4 编写二分法求方程实根的减半递推算法(用 C/C++ 描述)。

解：算法的 C++ 描述如下：

```
//ch1_3.cpp
#include<iostream>
#include<cmath>
using namespace std;
double root(double a,double b,double eps,double (*f)(double))
{ double f0,f1,c;
  f0=(*f)(a);
  while (fabs(a-b)>=eps)
  { c=(a+b)/2; f1=(*f)(c);
    if (f1+1.0==1.0) return(c);
    if (f0*f1>0) a=c;              //取区间后半部分
    else b=c;                      //取区间前半部分
  }
  c=(a+b)/2;
  return(c);
}
```

主函数例如下：

```
//主函数
```

```
int main()
{ double a=0.0, b=3.0;
  double f(double);
  cout<<root(a,b,0.00001,f)<<endl;
  return 0;
}
double f(double x)
{ return(x*x*x-x*x-1.0); }
```

运行结果如下：

```
1.46557
```

1.5 编写用回溯法求解皇后问题的算法(用 C/C++ 描述)。

解：算法的 C++ 描述如下：

```
//ch1_4.cpp
#include<iostream>
#include<iomanip>
#include<cmath>
using namespace std;
void queen(int n)
{ int i,j,k,jt,*q;
  q=new int[n];
  for (i=0; i<n; i++) q[i]=0;
  i=0; jt=1;
  cout<<n<<"queen problem"<<endl;
  while (jt==1)
  { if (q[i]<n)
    { k=0;
      while ((k<i)&&((q[k]-q[i])*(fabs(q[k]-q[i])-fabs(k-i)))!=0) k++;
      if (k<i) q[i]=q[i]+1;
      else
        { i=i+1;
          if (i>n-1)
            { for (j=0; j<n; j++) cout<<setw(5)<<q[j]+1;
              cout<<endl;
              q[n-1]=q[n-1]+1;
              i=n-1;
            }
        }
    }
  else
      { q[i]=0; i=i-1;
        if (i<0) { cout<<endl; delete q; return; }
        q[i]=q[i]+1;
      }
  }
}
```

主函数例如下：

```
//主函数
int main()
{ int n;
  cout<<"input n:";
  cin>>n;
  queen(n);
  return 0;
}
```

运行结果如下：(带有下划线的为键盘输入)

```
input n:4
4queen problem
2  4  1  3
3  1  4  2
```

1.6 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 C/C++ 描述)，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。

具体比较过程如下。

首先将 12 个元素分成以下三组：

第一组为 A(1)，A(2)，A(3)，A(4)四个元素；

第二组为 A(5)，A(6)，A(7)，A(8)四个元素；

第三组为 A(9)，A(10)，A(11)，A(12)四个元素。

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,可以写出算法的C++描述如下:

```
//ch1_5.cpp
#include<iostream>
using namespace std;
int a12(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]中有坏且重,或a[5],a[6],a[7],a[8]中有坏且轻
  if (a[1]+a[2]+a[6]==a[3]+a[4]+a[5])             //a[7],a[8]中有坏且轻
```

```
    if (a[1]==a[7]) return(-8);                          //a[8]坏且轻
    else return(-7);                                     //a[7]坏且轻
  else if (a[1]+a[2]+a[6]>a[3]+a[4]+a[5])//a[1],a[2]中有坏且重,或 a[5]坏且轻
    if (a[1]==a[2]) return(-5);                          //a[5]坏且轻
    else if (a[1]>a[2]) return(1);                       //a[1]坏且重
    else return(2);                                      //a[2]坏且重
  else //a[3],a[4]中有坏且重,或 a[6]坏且轻
    if (a[3]==a[4]) return(-6);                          //a[6]坏且轻
      else if (a[3]>a[4]) return(3);                     //a[3]坏且重
      else return(4);                                    //a[4]坏且重
    else//a[1],a[2],a[3],a[4]中有坏且轻,或 a[5],a[6],a[7],a[8]中有坏且重
      if (a[1]+a[2]+a[6]==a[3]+a[4]+a[5])                //a[7],a[8]中有坏且重
      if (a[1]==a[7]) return(8);                         //a[8]坏且重
      else return(7);                                    //a[7]坏且重
  else if (a[1]+a[2]+a[6]>a[3]+a[4]+a[5])//a[3],a[4]中有坏且轻,或 a[6]坏且重
      if (a[3]==a[4]) return(6);                         //a[6]坏且重
      else if (a[3]>a[4]) return(-4);                    //a[4]坏且轻
      else return(-3);                                   //a[3]坏且轻
  else //a[1],a[2]中有坏且轻,或 a[5]坏且重
      if (a[1]==a[2]) return(5);                         //a[5]坏且重
      else if (a[1]>a[2]) return(-2);                    //a[2]坏且轻
      else return(-1);                                   //a[1]坏且轻
}
```

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。

主函数例如下:

```
//主函数
int main()
{ int a[13]={0,5,5,5,5,5,5,5,4,5,5,5,5};
  cout<<"k="<<a12(a)<<endl;
  return 0;
}
```

运行结果如下:

```
k=-8      //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。下面举一个例子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：(1) 面向过程的 C++ 描述。

```
//ch2_02.cpp
#include<iostream>
#include<iomanip>
using namespace std;
template<typename T>
void invsl(int n, T a[])
{ int k;
  T t;
  for (k=1; k<=n/2; k++)
      { t=a[k-1]; a[k-1]=a[n-k]; a[n-k]=t; }
  return;
}
```

主函数例如下：

```
//主函数
int main()
```

```
{ int a[10]={1,2,3,4,5,6,7,8,9,10};
  int k;
  for (k=1; k<=10; k++) cout<<setw(5)<<a[k-1];
  cout<<endl;
  invsl(10,a);
  for (k=1; k<=10; k++) cout<<setw(5)<<a[k-1];
  cout<<endl;
  return 0;
}
```

(2) 面向对象的 C++ 描述。

```
//ch2_2.cpp
#include<iostream>
using namespace std;
template<class T>                    //模板声明,数据元素虚拟类型为 T
class sq_LList                       //顺序表类
{ private:                           //数据成员
    int mm;                          //存储空间容量
    int nn;                          //顺序表长度
    T * v;                           //顺序表存储空间首地址
 public:                             //成员函数
    sq_LList(){ mm=0; nn=0; return;}
    sq_LList(int);                   //建立空顺序表,申请存储空间
    void prt_sq_LList();             //顺序输出顺序表中的元素与顺序表长度
    int flag_sq_LList();             //检测顺序表的状态
    void ins_sq_LList(int, T);       //在表的指定元素前插入新元素
    void del_sq_LList(int);          //在表中删除指定元素
    void invsl();                    //逆转顺序表
    };

//建立空顺序表
template<class T>
sq_LList<T>::sq_LList(int m)
{ mm=m;                              //存储空间容量
  v=new T[mm];                       //动态申请存储空间
  nn=0;                              //顺序表长度为 0,即建立空顺序表
  return;
}

//顺序输出顺序表中的元素与顺序表长度
template<class T>
void sq_LList<T>::prt_sq_LList()
{ int i;
  cout<<"nn="<<nn<<endl;
  for (i=0; i<nn; i++) cout<<v[i]<<endl;
```