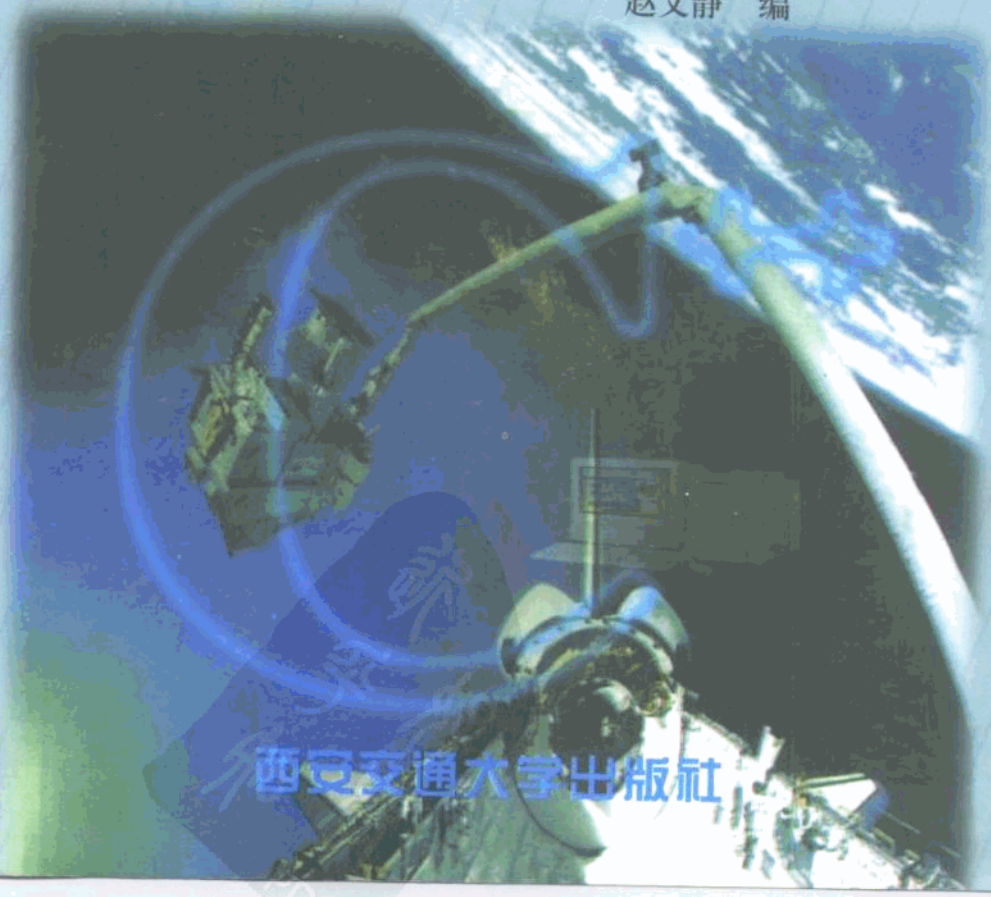
数据结构

——C++语言描述

赵文静 编



西安交通大学出版社

数据结构——C++语言描述
赵文静 编
西安交通大学出版社

前言

“数据结构”是计算机各专业的一门专业基础课，计算机科学各个领域及有关的应用软件都要用到各种数据结构。在我国计算机各专业的教学计划中，它是核心课程之一。本书介绍各种最常用的数据结构，阐明各种数据结构内在的逻辑关系，讨论它们在计算机中的存储表示，以及在这些数据结构上的运算和实际的算法，并对算法的效率进行了简要的分析。

本书共分13章，第1章至第4章介绍几种常用的线性结构：线性表、堆栈、队列和串；第5章至第6章介绍线性结构的典型应用——排序及检索；第7章至第9章介绍具有广泛应用价值的树形结构及其重要应用；第10章至第11章介绍复杂的数据结构——图、稀疏矩阵、广义表等；最后一部分（第12章至第13章）介绍外存储器中的数据结构和文件组织及整序。

本书既注重原理又重视算法的实现，配有大量的例题，均给出用C++语言实现的算法，并加上较详细的注释，以利于读者理解算法的基本思想。每章之后附有习题，以便读者进一步练习并检验学习效果。

本书的初稿于1992～1997年在我校作为计算机及应用专业“数据结构”课程的教材使用。经过不断改进并将原稿中用C语言描述算法改写为用C++描述算法，其目的是为了以抽象数据类型（ADT）讲述数据结构。本书的绝大多数例子都经上机用BORLAND C++ 4.5调试通过。

由于作者水平有限，书中错误在所难免，恳切希望读者批评指正。

赵文静

1997年于西安建筑科技大学

目 录

第1章 概 论

[内容提要]

本章将介绍有关数据和数据结构的概念，简述数据结构在计算机科学中的重要性，描述算法的语法及从时间和空间角度分析算法的方法。

[学习要点]

1. 熟悉各名词和术语的含义；掌握各种基本概念，特别是数据结构的三个方面以及它们的相互关系；熟悉数据结构的两大类型和4种基本的存储方法。
2. 熟悉 C++ 语言的书写规范。
3. 了解算法的5个要素(准则)及算法与程序的区别和联系。
4. 掌握估算算法的时间复杂度的方法。

1.1 为什么要学习数据结构

我们常常听到人们把计算机称作数据处理机，从而可见数据在计算机领域中的重要地位。我们在这里所说的数据就是对现实世界的事物采用计算机能够识别、存储和处理的形式所进行的描述。简言之，数据就是计算机化的信息。计算机一问世，数据作为计算机程序的处理对象也就随之产生。在计算机发展的初期，由于计算机主要用于数值计算，处理的是数值数据，而且数据量小、结构简单、形式统一。所以当时程序工作者把主要精力放在程序设计技巧上，而并不重视如何在计算机中组织数据。但是，随着计算机软件和硬件的发展，计算机应用领域的扩大，数据的概念也被大大推广了，越来越多的非数值数据需要处理。据统计，现在非数值计算占用了90% 以上的计算时间。计算机所处理的数据决不是杂乱无章的堆积，而是有着内在的联系，只有分析清楚它们的内在联系，对大量的数据才能进行有效处理。数据结构就是指数据间的相互关系。随着计算机应用的发展，不但处理的数据量越来越大，而且要处理的数据结构也更为复杂。

关于什么是“计算机科学”，从60年代以来一直有着争论，以 Wegnor 为代表的观点认为：计算机科学“包含一种关于计算图式的新的思想方法”。“工业革命中起核心作用的是‘能量’，在计算机革命中被‘信息’所取代”，他认为在计算机科学中具有核心作用的概念是“信息结构的转换”。计算机科学就是“一种关于信息结构转换的科学”。因而他认为构造有关信息结构转换的模型，对之进行研究，是计算机科学中带根本性的问题。而以 Knuth 为代表的另一种观点则认为：计算机科学“是算法的学问，算法是精确定义的一系列规则，指出怎样从给定的输入信息经过有限步骤产生所求的输出信息。‘算法’的特殊表示称为‘程序’，如同我们用‘数据’这个词来作为‘信息’的特殊表示一样”。我们且不去评论在计算机科学中到底是应该强调

“信息结构”(或“数据结构”)还是应该强调“算法”(或“程序”),但无疑“信息结构”和“算法”应该是计算机科学研究的基本课题,而且应该指出,“数据结构和算法之间存在着本质的联系,失去一方,另一方就没有什么意义。我们在谈到一种类型的数据结构时,总是离不开对这种类型数据结构需要施加的各种运算(又称操作,转换),例如,替换、插入、删除等等,而且只有通过对这些运算的算法的研究,才能更清楚地理解这种数据结构的意义和作用。反过来,当我们谈论一种算法时,也总是自然地联系到作为该算法处理对象和结果的数据。因此,在学习计算机科学中,数据结构的学习起着基础性的作用。

1.2 什么是数据结构

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。它是计算机程序加工的“原料”。例如,一个代数方程求解程序中所用的数据是整数和实数,而一个编译程序或文本编辑程序中使用的数据是字符串。随着计算机软、硬件的发展,计算机应用领域的扩大,数据的含义也随之拓广了。例如,当今计算机可以处理图象、声音等。因此它们也属于数据的范畴。

数据元素(Data Element)是数据的基本单位。有些情况下,数据元素也称为元素、结点、顶点、记录。有时一个数据元素可以由若干个数据项(也可称为字段、域)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。虽然至今还没有一个关于数据结构的标准定义,但它一般包括以下三方面的内容:

① 数据元素之间的逻辑关系,也称为数据的逻辑结构(Logical Structure)。

② 数据元素及其关系在计算机存储器内的表示,也称为数据的存储结构(Storage Structure)。

③ 数据的运算,即对数据施加的操作。

数据的逻辑结构是从逻辑关系上描述数据,它与数据的存储无关,是独立于计算机的。因此,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。数据的存储结构是逻辑结构在计算机存储器里的实现(亦称为映象),它是依赖于计算机的,对机器语言而言,存储结构是具体的,但我们只在高级语言的层次上来讨论存储结构。数据的运算是定义在数据的逻辑结构上的,每种逻辑结构都有一个运算的集合。例如,最常用的运算有:检索、插入、删除、更新、排序等。这些运算实际上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”,而无须考虑“如何做”。只有确定了存储结构之后,我们才考虑如何具体实现这些运算。本书中讨论的数据运算,均用面向对象的语言 C++ 以抽象数据类型描述相应的算法实现。

为了增加对数据结构的感性认识,下面举例说明上述概念。

例 1.1 学生成绩如表 1.1 所列:

表 1.1 学生成绩表

学号	姓 名	高等数学	英 语	物 理	程序设计	平均成绩
9705001	李力图	73	75	80	72	75
9705002	王大功	68	80	76	80	76
9705003	张小林	80	85	83	88	84
9705004	赵 刚	60	53	67	65	61
…	…	…	…	…	…	…

我们把这张学生成绩表称为一个数据结构，表中的每一行为一个数据元素（或结点，或记录），它由学号、姓名、各科成绩及平均成绩等数据项组成。该表中数据元素之间的逻辑关系是：对表中任一结点，与它相邻且在它前面的结点（亦称为直接前趋）（Immediate Predecessor）最多只有一个；与表中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趋，故称为开始结点；同时，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，表中“王大功”所在结点的直接前趋结点和直接后继结点分别是“李力图”和“张小林”所在的结点，上述结点间的关系构成了这张学生成绩表的逻辑结构。该表的存储结构则是指在计算机存储器中如何表示结点之间的这种关系，即表中的结点是顺序地邻接存储在一片连续的单元之中，还是用指针将这些结点链接在一起？在这张表中，可能要经常查看某一学生的成绩，当学生退学时要删除相应的结点，招收新生时要增加结点。究竟怎样进行查找、删除、插入，这就是数据的运算问题。搞清楚了上述三个问题，也就弄清了学生成绩表这个数据结构。

综上所述，我们可以将数据结构定义为：按某种逻辑关系组织起来的一批数据，按一定的存储表示方式把它们存储在计算机的存储器中，并在这些数据上定义了一个运算的集合，就叫做一个数据结构。

在不易产生混淆之处，我们常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

线性结构的逻辑待征是，有且仅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趋和一个直接后继。线性表就是一个典型的线性结构。本书第 2 章到第 6 章介绍的都是线性结构。

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趋和直接后继。本书第 7 章到第 13 章讨论的数据结构都是非线性结构。

数据的存储结构可用以下 4 种基本的存储方法得到：

(1) 顺序存储方法

该方法是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体现。由此得到的存储表示称为顺序存储结构（Sequential Storage Structure），通常顺序存储结构是借助于程序语言的向量来描述的。

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以通过某种线性化的方法来

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其物理位置上亦相邻,结点间的逻辑关系是由附加的指针字段表示的。由此得到的存储表示称为链式存储结构(Linked Storage Structure),通常要借助于程序语言的指针类型来描述。

(3) 索引存储方法

该方法通常是在存储结点信息的同时,还建立附加的索引表,索引表中的每一项称为索引项,索引项的一般形式是:(关键码,地址),关键码是能够唯一标识一个结点的那些数据项。若每个结点在索引表中都有一个索引项,则该索引表称为稠密索引(Dense Index),若一组结点在索引表中只对应一个索引项,则该索引表称之为稀疏索引(Sparse Index)。稠密索引中索引项的地址指示结点所在的存储位置,而稀疏索引中索引项的地址则指示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码直接计算出该结点的存储地址。

上述四种基本的存储方法也可以组合起来对数据结构进行存储映象。同一种逻辑结构采用不同的存储方法,可以得到不同的存储结构。选择何种存储结构来表示相应的逻辑结构,主要是使其运算方便并根据算法的时空要求来具体确定。

应当指出的是,很多教科书上是将数据的逻辑结构和数据的存储结构定义为数据结构,而将数据的运算定义为数据结构上的操作。但是,无论怎样定义数据结构,都应该将数据的逻辑结构、数据的存储结构及数据的运算这三方面看成一个整体,希望读者学习时,不要孤立地去理解一个方面,而要注意它们之间的联系。

正是因为存储结构是数据结构不可缺少的一个方面,所以我们常常将同一逻辑结构的不同存储结构冠以不同的数据结构名称来标识它们。例如,线性表是一种逻辑结构,若采用顺序方法的存储表示,则称该结构为顺序表;若采用链接方法的存储表示,则称该结构为链表;若采用散列方法的存储表示,则可称其为散列表。

同理,由于数据的运算也是数据结构不可分割的一个方面,所以给定了数据的逻辑结构和存储结构,若定义的运算集合及其运算的性质不同,也可能导致完全不同的数据结构。例如,若对线性表上的插入、删除运算限制仅在表的一端进行,则称该线性表为栈;而插入限制在表的一端进行,删除限制在表的另一端进行的线性表称之为队列。更进一步,若线性表采用顺序表和链表作为存储结构,则对插入和删除运算做了上述限制之后可分别得到顺序栈和链栈,顺序队列和链队列。

1.3 算法的描述

因为数据的运算是通过算法描述的,所以讨论算法是数据结构课程的重要内容之一。通俗地讲,一个算法就是一种解题方法。更严格地说,算法是由若干条指令组成的有穷序列,它必须满足下述准则:

① 输入:具有零个或多个输入的外界量,它们是算法开始前对算法最初给出的量。

② 输出:至少产生一个输出,它们是同输入有某种关系的量。

③ 有穷性：每一条指令的执行次数必须是有限的。

④ 确定性：每条指令的含义都必须明确，无二义性。

⑤ 可行性：每条指令的执行时间都是有限的。

在一个算法中，有些指令可能是重复执行的，因此指令的执行次数可能远远大于算法中的指令条数，由有穷性可知，对于任何输入，一个算法在执行了有限条指令后一定要终止；又由可行性知道，一个算法必须在有限时间内完成。因此，一个程序如果对任何输入都不会陷入无限循环，则它就是一个算法。

算法的含义与程序十分相似，但二者是有区别的。一个程序不一定满足有穷性。例如操作系统，只要整个系统不遭破坏，它就永远不会停止，即使没有作业要处理，它仍处于一个等待循环中，以待新作业的进入。因此操作系统程序不是一个算法。另外，程序中的指令必须是机器可执行的，而算法中的指令则无此限制。但是一个算法若用机器可执行的语言来书写，则它就是一个程序。

一个算法可以用自然语言，数学语言或约定的符号语言来描述。在本书中我们采用了面向对象的语言 C++ 描述相应的算法实现，且尽量以抽象数据类型描述数据结构。

什么是抽象数据类型？抽象数据类型(Abstract Data Type 简称 ADT)是指一个数学模型以及定义在该模型上的一组操作。抽象数据类型的定义仅取决于它的一组逻辑特性，而与其在计算机内部如何表示与实现无关，即不论其内部结构如何变化，只要它的数据特性不变，都不影响其外部的使用。

抽象数据类型是具有下述性质的数据类型：

(1) 数据和对数据的处理功能(函数)是一个完整的整体。

在传统的程序设计方法中，总是把数据结构和对数据进行处理的函数分开考虑。例如，使用 C 语言设计堆栈的时候，总是先利用结构完成数据的定义，然后再以这个结构为参数写出完成 push 和 pop 处理的函数。

与这种考虑方法不同，在抽象数据类型中则把函数看成是附属于数据类型的。也就是说，一个数据类型的数据结构(内部结构)和与该类型相关的基本运算(功能)是一个整体。在 C++ 类中，这种方案是由成员函数实现的。

(2) 仅由接口(由外部所见到的功能)定义。

在使用 int 类型或是 double 类型等基本数据类型时，很少有人关心 int 类型是由多少位二进制补码表示的，double 类型的指数有多少个二进制位，尾数有多少个二进制位。人们所关心的仅仅是该类型所能处理的数值范围以及是否能表示小数等等数据自身的性质和作用，也就是只关心它的外部性质。与此相同，抽象数据类型也是与内部构造无关而仅仅由其接口(从外部所看到的功能)的性质表示的数据类型。总之，不是以“内部构造是什么形式的”，而是以“能够干什么”定义的。

(3) 数据处理的实现方法对外部来说是不可见的。

抽象数据类型的另一个重要性质是“信息隐蔽”。这是指从外部看不到所定义的功能的实现方法和内部的数据结构。

抽象数据类型的定义格式：

ADT <ADT_Name>

Data

数据结构描述

Operations

构造函数及其它操作

end ADT

例如,我们用 C++ 定义抽象数据类型 String 如下:

```
class String                          //字符串类名(ADT 名)String
{
  private
   char * buff;                        //字符串类的数据成员,字符串指针
  public:
   int length;                        //字符串类的数据成员,字符串长度值
   String();                                  // 构造函数
   String(char * );                           // 构造函数
   ~String();                                  // 析构函数
   char& at(int);                             // 求指定位置处的字符
   void printString();                       // 成员函数,打印字符串值
   String& operator=(const char * );          // 赋值号重载成员函数
   String& operator=(const String&);          // 赋值号重载成员函数
   String& operator+(const char * );          // + 号重载成员函数
   String& operator+(String);                  // + 号重载成员函数
   friend  int operator>(String&,String&);   // > 号重载友元函数
   friend  int operator<(String&,String&);   // < 号重载友元函数
   friend  int operator==(String&,String&); // == 号重载友元函数
};
```

1.4 算法分析

求解同一个问题,可以有许多不同的算法,那么如何来评价这些算法的好坏呢?

显然,首先选用的算法应该是"正确的"。此外,主要考虑如下 3 点:

① 执行算法所耗费的时间;

② 执行算法所耗费的存储空间,其中主要考虑辅助存储空间;

③ 算法应易于理解,易于编码,易于调试等等。

当然我们希望选用一个所占存储空间小、运行时间短、其它性能也好的算法。然而,实际上很难做到十全十美。原因是上述要求有时相互抵触。要节约算法的执行时间往往要以牺牲更多的空间为代价;而为了节省空间又可能要以更多的时间作代价。因此我们只能根据具体情况有所侧重。若该程序使用次数较少,则力求算法简明易懂,易于转换为上机的程序。

对于反复多次使用的程序,应尽可能选用快速的算法。若待解决的问题数据量极大,机器的存储空间较小,则相应算法主要考虑如何节省空间。

一个算法所耗费的时间,应该是该算法中每条语句的执行时间之和,而每条语句的执行时

间是该语句的执行次数(也称为频度(Frequency Count))与该语句一次执行所需时间的乘积。但是当算法转换为程序之后,每条语句一次执行所需的时间取决于机器的指令性能、速度以及编译所产生的代码质量,这是很难确定的。因此我们假设每条语句的一次执行所需的时间均是单位时间,这样,一个算法的时间耗费就是该算法中所有语句的频度之和。于是,我们就可以独立于机器的软、硬件系统来分析算法的时间耗费。

一般情况下,算法中各语句重复执行的次数是问题规模 n 的某个函数 f(n),算法的时间量度记作:

$$T(n)=O(f(n)) \tag{1.1}$$

它表示随问题规模 n 的增大,算法执行时间的增长率和 f(n) 的增长率相同,称为算法的渐近时间复杂度(asymptotic time complexity),简称时间复杂度。

例 1.2 求两个 n 阶方阵的乘积 C=A×B,其算法的梗概如下:

```
  #define max 100
  void MATRIXMULT(int n,float a[max][max],float b[max][max],float c[max][max])
  |  int i,j,k; float x;
①  for(i=1;i<=n;i++){                                    n+1
②   for(j=1;j<=n;j++){                                   n(n+1)
③       x=0;                                             n²
④         for(k=1;k<=n;k++)                              n²(n+1)
⑤             x=x+a[i][k]*b[k][j];                       n³
⑥         c[i][j]=x;        }//end_for j                 n²
   }//end_for  i
  }
```

其中右边列出的是各语句的频度。语句①的循环控制变量 i 要增加到 n+1,测试 i>n 成立才会终止,故它的频度是 n+1,但是它的循环体却只能执行 n 次。语句②作为语句①循环体内的语句应该执行 n 次,但语句②本身要执行 n+1 次,所以语句②的频度是 n(n+1)。同理可得到语句③,④和⑤的频度分别是 n^2,$n^2(n+1)$,n^3。

该算法中所有语句的频度之和(即算法的时间耗费)为:

$$T(n)=2n^3+3n^2+2n+1 \tag{1.2}$$

由此可知,算法 MATRIXMULT 的时间耗费 T(n)是矩阵阶数 n 的函数:例如,算法 MATRIXMULT 的时间复杂度 T(n)如(1.1)式所示,当 n 趋向无穷大时,显然有

$$\lim_{n\to\infty}T(n)=\lim_{n\to\infty}\frac{2n^3+3n^2+2n+1}{n^3}=2$$

这表明,当 n 充分大时,T(n)和 n^3 之比是一个不等于零的常数,即 T(n)和 n^3 是同阶的,或说 T(n)和 n^3 的数量级相同,记作 $T(n)=O(n^3)$。我们称 $T(n)=O(n^3)$是算法 MATRIXMULT 的渐近时间复杂度。其中记号"O"是数学符号,其严格的数学定义是:

若 T(n)和 f(n)是定义在正整数集合上的两个函数,当存在两个正的常数 c 和 n_0,使得对所有的 $n\geqslant n_0$,都有 $T(n)\leqslant cf(n)$成立,则 $T(n)=O(f(n))$。

当我们评价一个算法的时间性能时,主要标准是算法时间复杂度的数量级,即算法的渐近时间复杂度。例如,设有两个算法 A_1 和 A_2,求解同一问题,它们的时间复杂度分别是 $T_1(n)$

$=100n$，$T_2(n)=n^2$，当输入量 $n<100$ 时，有 $T_1(n)>T_2(n)$，后者花费的时间较少。但是，随着问题规模 n 的增大，两个算法的时间开销之比 $n^2/100n$ 亦随着增大。也就是说，当问题规模较大时，算法 A_1 比算法 A_2 要有效得多，它们的渐近时间复杂度 $O(n)$ 和 $O(n^2)$，正是从宏观上评价了这两个算法在时间方面的质量。因此，在算法分析时，往往对算法的时间复杂度和渐近时间复杂度不予区分，而经常是将渐近时间复杂度 $T(n)=O(f(n))$ 简称为时间复杂度，其中的 $f(n)$ 一般是算法中频度最大的语句频度。例如，算法 MATRIXMULT 的时间复杂度一般是指 $T(n)=O(n^3)$，这里的 $f(n)=n^3$ 是该算法中语句⑤的频度。

有时，算法的时间耗费（执行频度最大的语句）还随问题的输入数据集的不同而不同。

例 1.3 冒泡排序

```
void bubsort(int a[maxsize],int n)
{ int t,j,i=1; bool exc=true;
while(i<=n&&exc){
        exc=false;
        for(j=1;j<=n-i;j++)
            if(a[j]>a[j+1]){ t=a[j]; a[j]=a[j+1]; a[j+1]=t; exc=true; }
        i++;
        }
}
```

此例中，执行频度最大的语句是“比较相邻两个整数”，当 a 中初始输入序列自小至大有序，则“比较相邻两个整数”执行次数为 $n-1$；当 a 中初始输入序列为自大至小有序，则“比较相邻两个整数”执行次数为 $n(n-1)/2$。对这类算法的分析，一种解决的办法是计算它的平均值，即考虑它对所有可能的输入数据集的期望值，此时相应的时间复杂度为算法的平均时间复杂度。例 1.3 中假设 a 的初始输入数据可能出现 n! 种排列情况的概率相等，则冒泡排序的平均时间复杂度 $T_{avg}(n)=O(n^2)$，然而，在很多情况下，各种输入数据集出现的概率难以确定，算法的平均时间复杂度也就难以确定。因此，另一种更可行更常用的办法是讨论算法在最坏情况下的时间复杂度，即分析最坏情况以估算算法执行时间的上界，例如冒泡排序的最坏情况是 a 中初始输入序列为自大至小有序，则冒泡排序算法在最坏情况下的时间复杂度为 $T(n)=O(n^2)$。在本书以后各章中讨论的时间复杂度，除特别指明外，均指最坏情况下的时间复杂度。

按数量级递增排列，常见的时间复杂度有常数阶 $O(1)$，对数阶 $O(\log_2{}^n)$，线性阶 $O(n)$，线性对数阶 $O(n\log_2{}^n)$，平方阶 $O(n^2)$，立方阶 $O(n^3)$，…，k 次方阶 $O(n^k)$，指数阶 $O(2^n)$。即

$$O(1)\leqslant O(\log_2{}^n)\leqslant O(n\log_2{}^n)\leqslant O(n^2)\leqslant O(n^3)\leqslant\cdots\leqslant O(n^k)\leqslant O(2^n) \tag{1.3}$$

显然，时间复杂度为指数阶 $O(2^n)$ 的算法效率极低，在 n 值稍大时就无法应用。

类似于算法的时间复杂度，本书中以空间复杂度（space complexity）作为算法所需存储空间的量度，记作

$$S(n)=O(f(n)) \tag{1.4}$$

其中 n 为问题的规模（或大小）。一个上机执行的程序除了需要存储空间来寄存本身所用

指令、常数、变量和输入数据外，也需要一些对数据进行操作的工作单元和存储一些为实现计算所需信息的辅助空间。如果输入数据所占空间只取决于问题本身，和算法无关，则只需要分析除输入和程序之外的额外空间；否则应同时考虑输入本身所需空间（和输入数据的表示形式有关）。若额外空间相对于输入数据量来说是常数，则称此算法为原地工作，第5章讨论的有些排序算法就属于这类。如果所占空间量依赖于特定的输入，则除特别指明外，均按最坏情况来分析。

习　题

1.1　简述下列术语：数据，数据元素，数据结构，存储结构，线性结构，非线性结构。

1.2　设n为正整数，利用大"O"记号将下列程序段的执行时间表示为n的函数。

```
① i=1,k=100;
   while(i<n){ k=k+1;
               i+=10;
             }
③  i=1;  j=0;
while(i+j<=n)
  if(i>j)  j++;
   else  i++;
④ x=n; {n是不小于1的常数}
   y=0;
   while(x>=(y+1)*(y+1)) y++;
```

```
② for(i=1; i<=n; i++)
     for(j=1; j<=i; j++)
       for(k=1; k<=j; k++)
         x=x+delta;
⑤  x=91;  y=100;
while(y>0)
   if(x>100) { x-=10;
   y--;
       }
else x++
```

1.3　设有数据结构(D,R)，其中：

$D=\{d_1,d_2,\cdots,d_6\}$　　$R=\{r\}$

$r=\{<d_1,d_3>,<d_1,d_5>,<d_2,d_4>,<d_2,d_5>,<d_3,d_6>,<d_5,d_6>\}$

试按图论中的画法惯例画出其逻辑结构图。

1.4　试写一算法，自大至小依次输出顺序读入的3个整数x,y和z的值。

1.5　已知k阶裴波那契序列的定义为

$f_0=0,\ f_1=0,\cdots,\ f_{k-2}=0,\ f_{k-1}=1;$

$f_n=f_{n-1}+f_{n-2}+\cdots+f_{n-k},\ n=k,k+1,\cdots$

试编写求k阶裴波那契序列的第m项值的函数算法，k和m均以参数的形式在参 量表中出现。

1.6　试编写算法计算$i!*2^i$的值并存入数组a[arrsize]的第i个分量中$i=1,2,\cdots,n$。假设计算机中允许的整数最大值为maxint，则当n>arrsize或对某个$k(1\leqslant k\leqslant n)$，使$k!*2^k>$maxint时，应按出错处理。可有下列3种不同的处理方式：

(1) 用ERROR语句终止执行并报告错误；

(2) 用布尔函数实现算法以区别正确返回或错误返回；

(3) 在参数表中设置一整型变量以区别正确返回或某种错误返回。

试讨论这3种方法各自的优缺点，并以你认为较好的方式实现之

1.7 试编写算法求一元多项式 $p_n(x)=\sum_{i=0}^{n}a_ix^i$ 的值 $P_n(x_0)$，并确定算法中每一语句的执行次数和整个算法的时间复杂度。注意：本题中的输入为 $a_i(i=0,1,\cdots,n)$，x_0 和 n，输出为 $P_n(x_0)$。通常算法的输入和输出可采用下列 3 种方式之一：

(1) 通过 cin 或 scanf 和 cout 或 printf；

(2) 通过参数表中的参数显式传递；

(3) 通过全局变量隐式传递。

试讨论这 3 种方法的优缺点，并在本题算法中以你认为较好的 1 种方式实现输入和输出。

1.8 按增长率由小至大的顺序排列下列各函数：

$2^{100},(3/2)^n,(2/3)^n,(4/3)^n,n^n,n^{3/2},n^{2/3},n!,n,\log_2{}^n,n/\log_2{}^n,n\log_2{}^n$

1.9 已知有实现同一功能的两个算法，其时间复杂度分别为 $O(2^n)$ 和 $O(n^{10})$，假设计算机可连续运算的时间为 10^7 秒(100 多天)，又每秒可执行基本操作(根据这些操作来估算 算法时间复杂度)10^5 次。试问在此条件下，这两个算法可解问题的规模(即 n 值的范围)各为多少？哪个算法更适宜？请说明理由。

1.10 试设定若干 n 值，比较两函数 n^2 和 $50n\log_2{}^n$ 的增长趋势，并确定 n 在什么范围内，函数 n^2 的值大于 $50n\log_2{}^n$ 的值。

1.11 试用数学归纳法证明：

(1) $\sum_{i=1}^{n}\frac{n(n+1)(2n+1)}{6}$

(2) $\sum_{i=1}^{n}X^i=(X^{n+1}-1)/(x-1)\qquad(x\neq 1,n\geqslant 0)$

第2章 线性表

[内容提要]

本章的基本内容是:线性表的逻辑结构定义和各种存储结构的描述方法;在线性表的两类存储结构(即顺序表和链表)上如何实现基本运算。

[学习要点]

1. 了解线性表的逻辑结构特性,当数据元素之间存在着线性关系时称之为线性表。在计算机中,用顺序存储方法来表示这种线性关系,则得到线性表的顺序存储结构(即顺序表);用链接存储方法来表示这种线性关系,则得到线性表的链式存储结构(即链表)。

2. 熟练掌握顺序表和链表的描述方法、特点及有关概念。本书中顺序表是用一个一维数组(即向量)和一个指示当前表中结点数目的指针 size 来描述的,C 中向量的下界是 0,如果当前表的长度是 size,则表元素存放在下标 0～size－1,但在顺序表的操作中仍然按习惯将表元素位置称为 1～size。

链表有两大类:一类是动态链表,它是由指针类型描述的,其特点是结点空间的分配和释放都是动态执行的;另一类是静态链表,它是用游标来模拟指针实现的,但用来存放表结点的空间仍是一个静态分配的向量空间。两类链表都可以有单链表,循环链表及双链表等多种形式,必须熟悉这些链表各自的特点。通常,一个链表是由一个头指针唯一确定的,但单(向)循环链表用一个尾指针表示会给某些运算带来方便。此外,还应注意链表中头指针,头结点和首元结点的区别。

应深刻理解动态链表中指针变量 p 和结点变量 * p 的对应关系。动态链表中结点 * p 的直接后继结点是(* p).next(或 p->next),静态链表中结点 sa[i]的直接后继结点是 sa[sa[i].next]。

3. 重点掌握顺序表上的插入和删除算法以及动态链表的建立,查找,插入和删除算法。

4. 掌握从时间和空间的角度综合比较顺序表和链表的不同特点及其适用场合。

[学习难点]

1. 动态链表的算法实现。

2. 静态链表的存储管理及算法实现。

2.1 线性表的定义及其基本运算

2.1.1 线性表的逻辑结构定义

线性表是最简单和最常用的一种数据结构。线性表的例子不胜枚举。例如,英文字母表,(A,B,…,Z)是一个线性表,表中的每一个英文字母是一个数据元素,又如,一副扑克牌的点数