1 绪论

随着计算机技术的飞速发展，计算机应用的范围越来越广泛，从最初的数值计算，发展到现在的数据处理、自动控制、信息处理、人工智能、情报检索和办公自动化等众多非数值计算领域。所处理的数据也从简单的数值发展到复杂的文字、图形、图像、音频、视频和动画等具有结构的数据。因此，要想高效地处理这些数据，必须深入研究数据本身的特性、数据之间的关系，以及如何有效地将数据存储在计算机内，这正是数据结构这门课程所要研究的主要问题。本章主要介绍数据结构的基本概念，数据的逻辑结构、存储结构及关系，算法及算法时间复杂度的分析。

1.1 数据结构的课程地位及研究内容

数据结构是计算机及相关专业的一门专业基础课，是介于数学、计算机硬件和计算机软件之间的一门核心课程，是程序设计的后续课程，同时也是编译原理、操作系统、数据库等课程的基础。本书较系统地介绍了软件设计及系统开发中经常使用的数据结构及相应的存储结构和算法、常用的查找和排序技术，并对各种结构和技术进行了比较。

在计算机发展的初期，人们使用计算机的目的主要是处理数值计算问题。当我们使用计算机来解决一个具体问题时，一般需要经过下列几个步骤：首先要从该具体问题抽象出一个适当的数学模型，然后设计或选择一个解此数学模型的算法，再编写程序进行调试、测试，最后运行程序并得到答案。例如，求解梁架结构中应力数学模型的线性方程组，可以使用迭代算法来实现；求解鸡兔同笼问题，可以通过二元一次方程组来实现。

由于计算机应用初期所涉及的运算对象只是简单的整型、实型或布尔型数据，所以程序设计者的主要精力集中于程序设计的技巧上，而忽略了数据结构。随着计算机应用领域的扩大

和软、硬件的发展，非数值计算问题显得越来越重要。据统计，当今处理非数值计算问题占用了 90%以上的机器时间。这类问题涉及的处理对象不再是简单的数据类型，其形式更加多样、结构更为复杂，数据元素之间的相互关系一般无法直接用数学方程式加以描述。因此，解决这类问题的关键不再是数学分析和计算方法，而是要设计出合适的数据结构，以便有效地解决问题。下面，我们先看几个例子。

例 1-1　公司员工信息管理系统。

某公司有一批员工，现需要用计算机来管理员工信息，要求能够完成以下操作：当招聘新员工时，能把员工信息添加进来；当有员工辞职时，能够删除该员工信息；能够修改员工信息；能够以某种方式检索员工信息。

分析：通过对以上问题的描述，我们可以把公司员工信息用表 1-1-1 表示出来。其中，每个员工的信息由员工号、姓名、性别、年龄等组成，员工数据按照一定的顺序线性排列。这就是解决该问题的模型（线性表）。有了模型以后，就可以围绕该模型设计算法，即实现员工信息的添加、修改、删除、检索等操作。

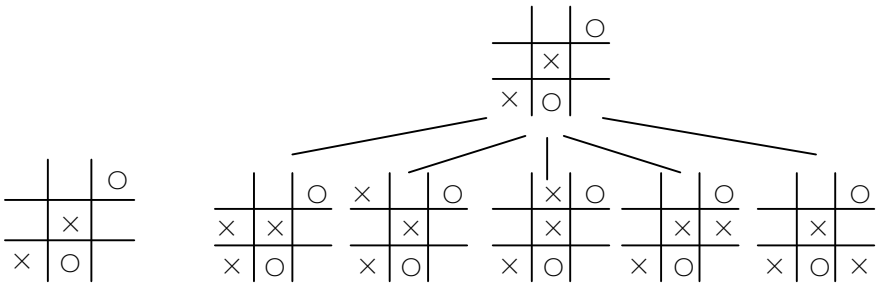
表 1-1-1　员工信息表

员工号	姓名	性别	年龄	住址	电话	所属部门
01002	王清	男	25	南京路 10 号	3562	财务
01003	李力	女	28	甘肃路 6 号	5673	总务
01004	张娟	女	30	杭州路 25 号	2345	经理办公室
01005	张爱民	男	35	洛阳路 12 号	2436	销售
……	……	……	……	……	……	……

类似地还有学籍管理系统、飞机订票系统、图书管理系统、学生选课系统等，它们都有一个共同点，即被处理的对象之间具有简单的线性关系，这就是一类数据结构——线性结构。

例 1-2　计算机和人对弈问题。

计算机之所以能和人对弈是因为有人将对弈的策略事先已存入计算机，包括对弈过程中所有可能的情况以及响应的策略。由于对弈的过程是在一定的规则下随机进行的，所以，为使计算机能灵活对弈就必须对对弈过程中所有可能发生的情况以及对应的策略都考虑周全。因此在对弈问题中，计算机操作的对象是对弈过程中可能出现的棋盘状态——称为格局。图 1-1-1（a）所示为井字棋的一个格局，根据比赛规则，通常一个格局可以派生出多个格局，图 1-1-1（b）所示为从图 1-1-1（a）派生出的 5 个格局，并且从每个新的格局又可以派生出 4 个可能出现的格局。因此，若将对弈从开始到结束的过程中所有可能出现的格局都画在一张图上，则可得到一棵倒长的“树”。树根是对弈开始之前的棋盘格局，而所有的叶子就是可能出现的结局，对弈的过程就是从树根沿树杈到某个叶子的过程。树也是一种数据结构，常用于表达某些非数值计算问题。



（a）棋盘格局示例　　（b）对弈树的局部

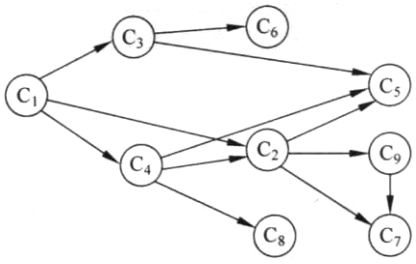
图 1-1-1　井字棋对弈树

例 1-3　教学计划编排问题

一个教学计划包含许多课程，在教学计划包含的许多课程之间，有些课程必须按规定的先后次序进行学习，有些则没有次序要求。也就是说，有些课程之间有先修和后修的关系。课程之间先修和后修的次序关系可用一个称作“图”的数据结构来表示，如图 1-1-2 所示，图中的每个顶点表示一门课程，如果从顶点 v_i 到 v_j 之间存在有向边$<v_i, v_j>$，则表示课程 i 必须先于课程 j 进行。如何在规定的年限内修完所有的课程，在哪个学期安排哪个课程，就是本例要进行的操作。

课程编号	课程名称	选修课程
C_1	计算机导论	无
C_2	数据结构	C_1，C_4
C_3	汇编语言	C_1
C_4	C 程序设计语言	C_1
C_5	计算机图形学	C_2，C_3，C_4
C_6	接口技术	C_3
C_7	数据库原理	C_2，C_9
C_8	编译原理	C_4
C_9	操作系统	C_2

（a）计算机专业的课程设置



（b）表示课程之间优先关系的有向图

图 1-1-2　教学计划编排问题的数据结构

由以上几个例子可见，描述非数值计算问题的数学模型不再是数学方程，而是诸如线性表、树、图之类的数据结构。因此，数据结构课程是研究非数值计算的程序设计问题中计算机处理对象以及它们之间关系和操作的学科。它主要研究：①数据元素之间的逻辑关系——数据的逻辑结构；②数据元素及关系在计算机内的表示——数据的存储结构；③数据的操作及实现。

学习数据结构的目的是了解和掌握计算机处理对象的特性，将实际问题中所涉及的处理对象在计算机中表示出来并对它们进行处理。同时，通过算法训练来提高学生的思维能力，通过程序设计的技能训练来促进学生的综合应用能力和专业素质的提高。

1.2 基本概念和术语

在系统地学习数据结构知识之前，首先对一些基本概念和术语赋予确切的定义。

1. *数据*（data）

数据是所有能被输入计算机、且能被计算机处理的符号的集合；是计算机加工处理的对象。它可以是数值型数据（整数、实数、复数、双精度数等），也可以是非数值型数据（字符、字符串、声音、图像、图片、语音等）。

2. *数据元素*（data element）

数据元素是数据的基本单位。在计算机程序中通常作为一个整体来考虑和处理。在不同的条件下，数据元素又可称为元素、结点、顶点、记录等。例如，学生信息检索系统中学生信息表中的一个记录、教学计划编排问题中的一个顶点等，都被称为一个数据元素。

一个数据元素可由若干个数据项组成。

3. *数据项*（data item）

数据项是组成数据元素的单位，是数据的不可分割的最小单位。图 1-2-1 以形象的形式指出了数据元素和数据项的关系。

学　号	姓　名	英　语	高等数学	C 语言程序设计
20050001	张　三	85	73	89
20050002	李　四	79	84	92
20050003	王　五	68	58	75

一个数据项　　一个数据元素

图 1-2-1　学生成绩表

4. *数据结构*（Data Structure）

数据结构指的是数据之间的相互关系，即数据的组织形式，是互相之间存在着一种或多种关系的数据元素的集合。数据结构包括三个方面的内容：数据的逻辑结构、存储结构和对数

据的运算（或操作）。

（1）数据的逻辑结构（Logical Structure）

在任何问题中，数据元素之间都不会是孤立的，在它们之间都存在着这样或那样的关系，这种数据元素之间存在的关系称为数据的逻辑结构。数据的逻辑结构可以看作是从具体问题抽象出来的模型，它与数据的存储无关。逻辑结构可归结为以下四种基本类型：

1）集合结构：结构中的数据元素之间除了“同属于一个集合”的关系外，别无其他关系。可以用一些离散的圆圈来表示集合的元素。比如五个元素的集合如图 1-2-2 所示。

2）线性结构：结构中的数据元素之间存在一对一的关系。图 1-2-3 是五个元素的线性结构。

3）树型结构：结构中的元素之间存在一对多的关系。图 1-2-4 是七个元素的树型结构。

4）图状结构：结构中的元素之间存在多对多的关系。该结构也称作网状结构。图 1-2-5 是五个元素的图状结构。

其中树型结构和图状结构是典型的非线性结构。

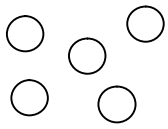


图 1-2-2　集合

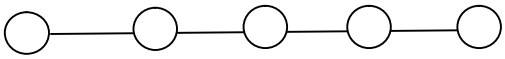


图 1-2-3　线性结构

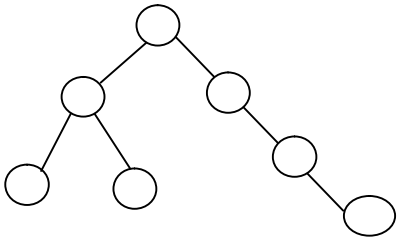


图 1-2-4　树型结构

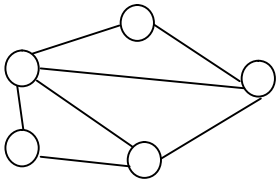


图 1-2-5　图状结构

在形式上，数据结构通常可以采用一个二元组来表示：

Data_Structure=(D,R)

其中 D 是数据元素集合，R 是 D 中元素之间关系的集合。

例 1-4　有一种数据结构 Line=(D,R)，其中

D={1，2，3，4，5，6，7}

R={<3,7>，<7,1>，<1,5>，<5,2>，<2,4>，<4,6>}

这种数据结构的表示如图 1-2-6 所示，显然是一个线性结构。

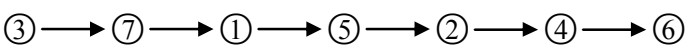


图 1-2-6　线性结构

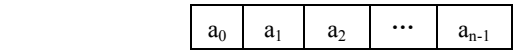
（2）数据的存储结构（Storage Structure）

数据的存储结构是指数据的逻辑结构用计算机语言的实现，即数据元素及其关系在计算机存储器内的表示，也称为数据的物理结构。

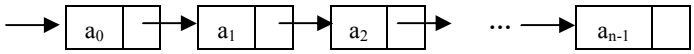
数据的存储可以采用以下四种基本的存储结构：顺序存储、链式存储、索引存储和散列存储。

1）顺序存储：是把逻辑上相邻的结点存储在物理位置相邻的存储单元里，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，见图 1-2-7（a）。

2）链式存储：结点间的逻辑关系由附加的指针字段表示，它不要求逻辑上相邻的结点在物理位置上亦相邻。由此得到的存储表示称为链式存储结构，见图 1-2-7（b）。



（a）顺序存储结构



（b）链式存储结构

图 1-2-7　基本存储结构形式

3）索引存储：除建立存储结点信息外，还建立附加的索引表来标识结点的地址。

4）散列存储：根据结点的关键字直接计算出该结点的存储地址。

上述四种基本的存储方法，既可以单独使用，也可以组合起来对数据结构进行存储映像。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，要视具体要求而定，主要考虑运算方便及算法的时空要求。

（3）数据的运算

数据的运算即对数据施加的操作，如插入、删除、检索等。在数据结构中，这些运算需要通过算法来实现。

数据的逻辑结构、数据的存储结构及数据的运算这三方面是一个整体。孤立地去理解任一个方面，而不注意它们之间的联系是不可取的。最后通过例子来增加对数据结构三方面内容的感性认识。

有一张学生成绩表，记录了一个班的学生各门课的成绩，这个表就是一个数据结构，见表 1-2-1。

在表中，每一行记录了一个学生的学号、姓名和三门课的成绩及平均成绩，称为该数据结构中的一个元素或一个结点。对于整个表来说，只有一个开始结点（它的前面无记录）和一个终端结点（它的后面无记录），其他的结点则各有一个也只有一个直接前驱和直接后继（它的前面和后面均有且只有一个记录）。这几个关系就确定了这个表的逻辑结构（为线性结构）。那么我们怎样把这个表中的数据存储到计算机里呢？用高级语言如何表示各结点之间的关系呢？是用一片连续的内存单元来存放这些记录（如用数组表示）？还是随机存放各结点数据再

用指针进行链接呢？这就是存储结构的问题，我们将从高级语言的层次来讨论这个问题。最后，我们有了这张表（数据结构），肯定要用它，那么就是要对这张表中的记录进行查询、修改、删除等操作，对这个表可以进行哪些操作以及如何实现这些操作就是数据的运算问题。

表 1-2-1　学生成绩表

学　号	姓　名	数　学	英　语	数据结构	平均成绩
20021001	张三	88	90	92	90
20021002	李四	86	84	85	85
20021003	王五	93	92	94	93
20021004	赵六	89	93	91	91
20021005	丁一	92	88	87	89
⋮	⋮	⋮	⋮	⋮	⋮

数据结构课程主要研究数据的逻辑结构、相应的存储结构以及定义在它们之上的一组运算，并要求设计出相应的算法，同时还必须分析算法的效率。

5. 数据类型（Data Type）

数据类型是程序设计语言中所允许使用的变量种类。一个数据类型不仅定义了相应变量可以设定的值的集合，而且还规定了对变量允许进行的一组运算及其规则。例如，在 C 语言中定义的整型（int）是-32768～32767 范围内所有整数的集合，以及可以对整数进行的加、减、乘、除、求余等运算。对用户来说，只需了解整数的各种运算的抽象特性，而不必了解计算机实现这些计算的细节，这样就可以使用高级语言进行程序设计。所以通常可以把数据类型看作是程序设计语言中已经实现了的数据结构。

6. 抽象数据类型（Abstract Data Type，ADT）

抽象数据类型是指基于一类逻辑关系的数据类型以及定义在这个类型上的一组操作。抽象数据类型的定义取决于它的逻辑特性，而与在计算机内部如何表示和实现无关。在软件设计中，抽象数据类型的定义可以看作是描述问题的模型，它独立于具体实现，一般可以由元素、元素之间的关系及操作三种要素来定义。抽象数据类型的描述格式如下：

```
ADT 抽象数据类型名
{
    数据对象：<数据对象的定义>
    数据关系：<数据关系的定义>
    基本操作：<基本操作的定义>
}ADT 抽象数据类型名
```

其中数据对象和数据关系的定义用伪码描述。

例 1-5　抽象数据类型“字符串”的定义。

```
ADT String
{
数据对象：D={ai|ai∈Character，i=1，2，… n，n≥0}
```

数据关系：R={<a_{i-1},a_i>|a_{i-1},a_i∈D，i=2，…n}

基本操作：

StrCopy(&T,S)：将串 S 复制得到串 T。
StrCompare(S,T)：若 S>T 返回正数，若 S==T 返回 0，若 S<T 返回负数。
StrLength(S)：返回 S 中元素的个数，即串的长度。
ClearString(&S)：将 S 清为空串。
ConCat(&T,S_1,S_2)：用 T 返回由 S_1 和 S_2 连接而成的新串。
SubString(&Sub,S,pos,len)：用 Sub 返回串 S 的第 pos 个字符起长度为 len 的子串。
Index(S,T,pos)：若主串 S 中存在和串 T 值相同的子串，则返回它在主串 S 中第 pos 个字符之后第一次出现的位置，否则函数值为 0。
……
}ADT String

抽象数据类型的定义是软件设计者之间的接口。当应用程序中需要用到的各种基本数据结构都有设计好的、可重复使用的抽象数据类型时，以后的程序设计和程序维护将大大简化。

1.3 算法的描述和分析

瑞士计算机科学家 N.Wirth 指出“程序=数据结构+算法”。它描述了计算机程序是由组织信息的数据结构和处理信息的算法组成。二者相辅相成，不可分割，对实际问题的求解，就是选择一种好的数据结构，加上设计一个好的算法。

1.3.1 算法

讲算法之前先看一个问题的求解过程。

例 1-6 已知 n 个整数，求 n 个整数中的最大数。

这是一个非常简单的问题，下面给出求解过程：

1）将第一个数赋给 max；

2）初始化计数变量 i 为 1；

3）当 i<n 时执行以下内容：比较 a[i]与 max，若 a[i]大于 max，则将 a[i]赋给 max；i 自增 1，继续比较；

4）返回 max 的值。

这就是一个求最大数的算法。因此，算法是对特定问题求解过程的描述，是一个能够解决问题的、有具体步骤的方法，是指令的有限序列。

算法可以用伪码、自然语言和框图等形式描述，也可以用程序设计语言描述。从易于实现的角度考虑，通常用某种程序设计语言来描述算法，其优点是直接作为程序语句输入计算机后，计算机就能调用和运行，从而实现对问题的求解。对例 1-5 的算法，用程序设计语言描述如下：

```
int Max(int a[ ],int n)
{
    int i,max;
```

```
    max=a[0];
    for(i=1;i<=n;i++)
        if(a[i]>max)   max=a[i];
    return max;
}
```

一个算法应该具有下列特性：

（1）有穷性。对于任意一组合法输入值，在执行有穷步骤之后一定能结束，即算法中的每个步骤都能在有限时间内完成。

（2）确定性。对于每种情况下所应执行的操作，在算法中都有确切的规定，使算法的执行者或阅读者能明确其含义及如何执行。并且在任何条件下，算法都只有一条执行路径。

（3）可行性。算法中的所有操作都必须足够基本，都可以通过已经实现的基本操作运算有限次实现。

（4）有输入。作为算法加工对象的量值，通常体现为算法中的一组变量。有些输入量需要在算法执行过程中输入，而有的算法表面上可以没有输入，实际上已被嵌入算法之中。

（5）有输出。一个算法能产生一个或者多个输出，它是一组与“输入”有确定关系的量值，是算法进行信息加工后得到的结果，这种确定关系即为算法的功能。

算法的有穷性是算法和程序的分界点。程序并不要求在有限的步骤内或有限的时间内结束，比如操作系统，而算法却有这个要求；算法应该有输入和输出，主要是希望算法能够解决实际问题，能产生有意义的结果。

1.3.2 算法的设计要求

程序设计的实质是对确定的问题选择一种“好结构和好算法”，一个好的算法应符合以下算法设计要求：

（1）正确性：一个好的算法应当满足具体问题的要求。通常包含四层含义：不含语法错误；对几组输入数据运行正确；对精心选择的典型、苛刻而带有刁难性的几组输入数据运行正确；对一切合法的输入数据都能运行正确。

（2）可读性：算法主要是为了人的阅读与交流，其次才是为计算机执行，因此算法应该易于人理解；另一方面，晦涩难读的程序易于隐藏较多错误而难以调试。

（3）健壮性：当输入的数据非法时，一个好的算法应能做出适当反应或进行处理，而不会产生莫名其妙的输出结果。

（4）高效率与低存储量需求：一个好的算法应能够有效利用计算机的资源，执行时间短，存储空间小。

1.3.3 算法度量及分析

通常对于一个实际问题的解决，可以提出若干个算法，那么如何从这些可行的算法中找出最有效的算法呢？或者有了一个解决实际问题的算法，我们如何来评价它的好坏？这些问题

需要通过算法分析来确定。

人们一般从两个方面来衡量算法：一个是时间效率，即算法处理数据所花费的时间，用时间复杂度（Time Complexity）来表示；一个是空间效率，即算法所需的存储量的大小，用空间复杂度（Space Complexity）来表示。二者都应尽量得低，但二者往往不能同时兼顾，在目前计算机硬件价格大幅度下降的前提下，算法的时间效率应首先被考虑。

1. 时间复杂度

对于解决同一个问题的算法，执行时间短的显然比执行时间长的效率高，那么算法执行时间的长短如何度量呢？当一个算法被转换成程序运行时，许多因素会影响程序的运行时间，如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时，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。为此，可以撇开这些与计算机硬件软件有关的因素，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（通常用整数量 n 表示），或者说，它是问题规模的函数。

一个算法的时间复杂度是指该算法的运行时间与问题规模的对应关系。为了便于比较同一问题的不同算法，通常的做法是，从算法中选出一种对于所研究的问题来说是基本操作的元操作，以该基本操作重复执行的次数（也称为频度）作为算法的时间度量。一般情况下，算法中基本语句重复执行的次数 T(n)是问题的规模 n 的某个函数 f(n)，因此，算法的时间度量可记作

$$T(n)=O(f(n))$$

记号“O”读作“大 O”，它表示随着问题规模 n 的增大，执行时间的增长率和 f(n)的增长率相同。也称作算法的渐进时间复杂度（Asymptotic Time Complexity），简称为时间复杂度。

时间复杂度往往不是精确的执行次数，而是估算的数量级，它着重体现的是随着问题规模 n 的增大，算法执行时间的变化趋势。因此，“大 O 表示法”通常只关注 T(n)的最高阶，而忽略其低次项和常数项。例如，一个程序的实际执行时间为 $T(n)=2.7n^3+3.8n^2+5.3$，在这个多项式中随着 n 变大，n^3 项的成长比其他两项的成长速度快得多，因此这两项可以被忽略，可以说对于较大的 n 值，这个子程序基本上是一个 $2.7n^3$ 的处理程序。故 $T(n)=O(n^3)$。

例 1-7 求如下两个 N×N 矩阵相乘的算法的时间复杂度。

```
for(i=1;i<=n;++i)
    for(j=1;j<=n;++j)
        {
            c[i][j]=0;
            for(k=1;k<=n;++k)
                c[i][j]+=a[i][k]*b[k][j];
}
```

在以上代码中，乘法运算是“矩阵相乘问题”的基本操作。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，记作：

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

显然，基本操作重复执行次数和算法的执行时间是相同数量级的，多数情况下它是最深

层循环内的语句中的元操作。

例 1-8 分析下列三个程序段的时间复杂度。

（a）x=x+1;

（b）for (i=0; i<n; i++)　x=x+1;

（c）for (i=0; i<n; i++)

　　for (j=0; j<i; j++)

　　x=x+1;

分析：上述三个程序段的问题规模均为 n，算法中基本操作“x=x+1”的执行次数分别为 1、n、n(n+l)/2，相应的时间复杂度 T(n)分别为 O(1)、O(n)和 $O(n^2)$，分别称为常数阶、线性阶和平方阶。

常见的时间复杂度，按数量级递增排列依次为：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 $O(2^n)$，且 $O(1)\leqslant O(\log_2 n)\leqslant O(n)\leqslant O(n\log_2 n)\leqslant O(n^2)\leqslant O(n^3)\leqslant\ldots\leqslant O(n^k)\leqslant O(2^n)$ 。

说明：当 f(n)为对数函数、幂函数或它们的乘积时，算法的运行时间是可以接受的，称这些算法是有效算法；当 f(n)为指数函数或阶乘函数时，算法的运行时间是不可接受的，称这些算法是无效算法，因此应尽量少用指数阶的算法。

2. 空间复杂度

除了时间以外，空间代价也是程序员要经常考虑的计算机资源。近年来，在计算机运行速度提高的同时，其存储能力也大大增强了。虽然如此，可利用的磁盘或内存空间大小仍是对算法设计者的重要限制。类似于时间复杂度，一个算法的空间复杂度 S(n)定义为该算法所耗费的存储空间，记作

$$S(n)=O(f(n))$$

它也是问题规模（或大小）n 的函数。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、变量以外，也需要一些存储数据操作的工作单元和为实现计算所需信息的辅助空间。程序本身所有指令所占空间对不同算法来说一般不会有数量级的差别，在比较算法时可以不加考虑；若输入数据所占空间只取决于问题本身，而和算法无关，则在比较算法时也可以不加考虑。故对算法空间复杂度的分析通常只需要分析实现计算所需信息的辅助空间。

1.4 应用实践：学生管理系统登录模块设计

1. 实践目的

（1）掌握程序设计的三大结构：顺序结构、选择结构、循环结构。

（2）掌握结构体的定义和应用。

（3）掌握指针的定义和应用。

（4）掌握函数的编写和调用方法。

2. 实践内容

利用结构体和指针实现学生管理系统登录模块设计，要求先对多个用户信息进行初始化，用户信息包括账号和密码两部分，然后输入当前用户的账号和密码进行验证，正确则显示“登录成功!”，错误则显示“账号或者密码错误!”。

3. 代码清单

```
#include "stdio.h"
#include "string.h"
//用户结构体定义
typedef struct
{
    long id;
    char pwd[6];
}user;
user users[50];//数组定义，用来存储多个用户信息
void init(int n)//输入用户信息
{
    user *p;
    int i;
    p=users;//指针的应用
    printf("请管理员输入用户信息:\n");
    for(i=0;i<n;i++)
    {
        scanf("%ld,%s",&p[i].id,p[i].pwd);
    }
}
main()
{
    long stuid;
    char stupwd[6];
    int i;
    int n;
    printf("请输入用户数量：");
    scanf("%d",&n);
    init(n);//调用输入用户信息函数
    printf("请您输入账号和密码:");
    scanf("%ld,%s",&stuid,stupwd);
    for(i=0;i<n;i++)
    {
        if(stuid==users[i].id && strcmp(stupwd,users[i].pwd)==0)//验证
        {
            printf("登录成功！\n");
            break;
        }
    }
    if(i==5)
        printf("账号或者密码错误！");
}
```

4. 结果验证

上机实验结果如图 1-4-1 所示。

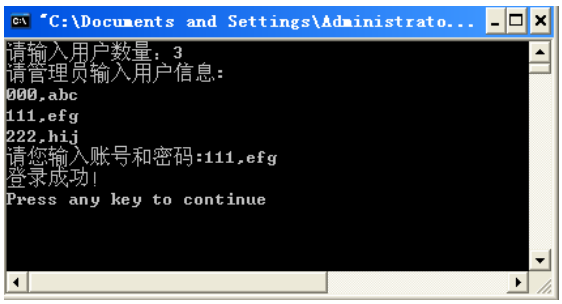


图 1-4-1 运行结果

小结

数据结构课程主要讨论非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作。本章主要介绍了数据、数据元素、数据结构等基本概念和抽象数据类型的定义、表示和实现方法，并详细阐述了算法设计的要求以及从时间和空间角度分析算法的方法。通过本章学习，要求同学们熟悉各名词、术语的含义，掌握数据的逻辑结构和存储结构之间的关系，掌握常见的四种逻辑结构和常用的四种存储结构；了解抽象数据类型相关内容；理解算法的特性及设计要求；掌握计算语句频度和估算算法时间复杂度的方法。

习题一

1．叙述下列概念：数据、数据结构、数据对象、存储结构、数据类型和算法。

2．试举一个数据结构的例子，叙述其逻辑结构、存储结构和运算三个方面的内容。

3．什么叫算法？评判算法的优劣有哪几种方法？

4．什么叫算法的时间复杂度？怎样表示算法的时间复杂度？

5．分析下列程序段，求其时间复杂度。

（1）i=1; k=0;

```
while (i<=n-1)
   { k=k*10*i;       i++;
   }
```

（2）i=1；j=0；

```
while(i+j<=n)
   {   if (i>j) j++;
```

```
        else        i++;
    }
```

（3）
```
for (i=0; i<n; i++)
    for (j=0; j<i; j++)
        for (k=0; k<j; k++)
            x=x+1;
```

（4）
```
x=n;                /* n>1 */
y=0;
while ((x>=(y+1)*(y+1))
    y=y+1;
```

6．已知输入 x，y，z 三个不相等的整数，试设计一个算法，使这三个数按从小到大的顺序输出，并考虑此算法元素的比较次数和元素的移动次数。

7．编写算法实现猴子吃桃问题：猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个；第二天早上将剩下的桃子吃掉一半，又多吃了一个；以后每天早上都吃了前一天剩下的一半零一个，到第 10 天早上想再吃时，见只剩下一个桃子了。求第一天共摘了多少个桃子。

8．编写算法：随机产生 n 个整数，然后用一种排序算法将它们从小到大排序。使用三种不同规模的数据进行试验：10 个，100 个，10000 个。