第 1 章

绪　论

计算机科学是一门研究数据表示和数据处理的科学。数据是计算机化的信息，它是计算机可以直接处理的最基本和最重要的对象。无论是进行科学计算、数据处理、过程控制，还是对文件进行存储和检索及应用数据库技术，在这些计算机应用领域中，它们都是对数据进行加工处理的过程。因此，要设计出一个结构好、效率高的程序，必须研究数据的特性及数据间的相互关系及其对应的存储表示，并利用这些特性和关系设计出相应的算法和程序。

数据结构也称为信息结构。著名计算机科学家 P.Wegner 认为：计算机科学就是“一种关于信息结构转换的科学”。“结构”一词可以当成动词看待，解释为“把某些成分（或称原子、元素、成员等）按一定规律组织在一起的过程和方式”；也可以作为名词，理解为“把某些成分按一定方式组织起来而形成的实体”。对于复杂程序，需要处理的数据往往很多，这些数据之间往往又有错综复杂的相互联系，因此要有效地存储这些数据及其相互关系，如何使它们能够在程序中有效地使用，以及如何把它们组织起来，这是我们要解决的一个最重要的问题。数据结构课程讨论的就是数据的组织问题。实际上，数据结构问题不仅与程序设计有关，而且与计算机科学的许多其他领域也都有密切的关系。学好这门课程对于深入地理解计算机科学，继续学习计算机科学涉及的其他课程是非常重要的。要搞好计算机方面的工作，无论是实际的应用开发工作，还是理论研究工作，本课程的内容都是必不可少的知识基础。

学习任何课程都必须了解它与本学科其他课程的联系，明确其在更广泛的学科领域整体中的位置，只有这样才能抓住学习的要领，理解课程内容的实质，对于数据结构也不例外。图 1-1 形象地显示了数据结构知识在应用计算机解决问题过程中的地位，以及它与计算机科学技术领域中其他课程之间的关系。

作为数据结构的基础，有算法分析与设计的理论和方法，以及关于数据模型的理论。实现数据结构需要采用某种描述语言（通常是某种程序设计语言），并遵循一定的设计方法。学习和研究数据结构的目的是解决问题。

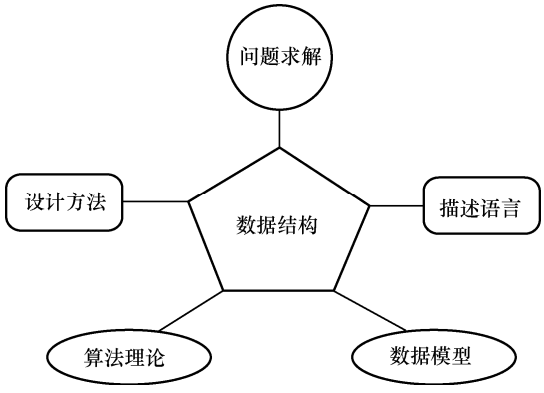

图 1-1　数据结构与其他课程的关系

1.1　数据结构的概念

数据结构是计算机科学与技术专业的基础课，是十分重要的核心课程。所有的计算机系

统软件和应用软件都要用到各种类型的数据结构。因此，要想更好地运用计算机来解决实际问题，仅掌握几种计算机程序设计语言是难以应付众多复杂的问题的。若要有效地使用计算机，充分发挥计算机的性能，还必须学习和掌握好数据结构的有关知识。打好“数据结构”这门课程的扎实基础，对于学习计算机专业的其他课程，如操作系统、编译原理、数据库管理系统、软件工程、人工智能等都是十分有益的。

1.1.1 为什么要学习数据结构

用计算机解决问题常常可以看成对实际问题的一种模拟，从这种观点出发，工作目标就是在计算机中建立一个与实际问题有着密切对应关系的模型，在这个模型中，计算机内存的数据表示了需要被处理的实际对象，包括其内在的性质和关系，处理这些数据的程序则模拟对象领域中的实际过程。最后，通过将计算机程序的运行结果在实际领域中给予解释，便得到了实际问题的解。

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，最后编写出程序进行调试、测试，直至得到最终的解答。例如，求解梁架结构中应力的数学模型的线性方程组，该方程组可以使用迭代算法来求解。

由于当时所涉及的运算对象是简单的整型、实型或布尔类型数据，所以程序设计者的主要精力是集中于程序设计的技巧上，而无须重视数据结构。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性问题占用了90%以上的机器时间。这类问题涉及的数据结构更为复杂，数据元素之间的相互关系一般无法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，才能有效地解决问题。

以下列举的就是这一类的具体问题。

【例 1-1】学生信息检索系统。当需要查找某个学生的有关情况的时候，或者想查询某个专业或年级的学生的有关情况的时候，只要我们建立了相关的数据结构，按照某种算法编写相关程序，就可以实现计算机自动检索。因此，可以在学生信息检索系统中建立一张按学号顺序排列的学生信息表和分别按姓名、专业、年级顺序排列的索引表。由这 4 张表构成的文件便是学生信息检索的数学模型，计算机的主要操作便是按照某个特定要求（如给定姓名）对学生信息文件进行查询。

诸如此类的还有电话自动查号系统、考试查分系统、仓库库存管理系统等。在这类文档管理的数学模型中，计算机处理的对象之间通常存在着一种简单的线性关系，这类数学模型可称为**线性的数据结构**。

【例 1-2】八皇后问题。在八皇后问题中，处理过程不是根据某种确定的计算法则，而是利用试探和回溯的探索技术求解。为了求得合理布局，在计算机中要存储布局的当前状态。从最初的布局状态开始，一步步地进行试探，每试探一步形成一个新的状态，整个试探过程形成了一棵隐含的状态树，如图 1-2 所示（为了描述方便，将八皇后问题简化为四皇后问题）。回溯法求解过程实质上就是一个遍历状态树的过程。在这个问题中所出现的树也是一种数据

结构，它可以应用在许多非数值计算的问题中。

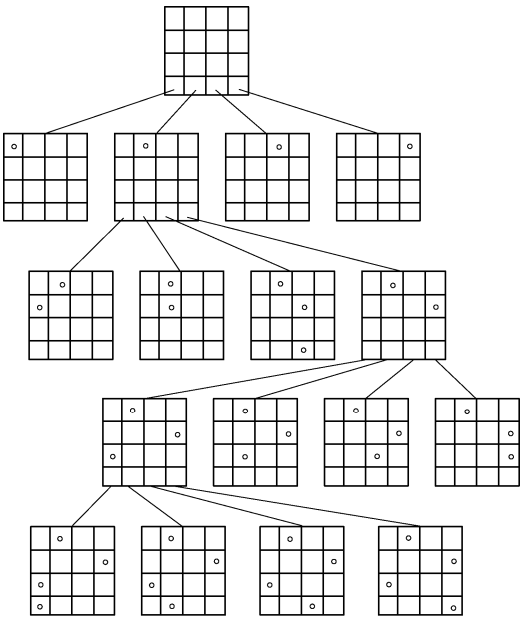

图 1-2　四皇后问题求解形成的树结构

【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，有些必须按规定的先后次序进行，有些则没有次序要求（见表 1-1），即有些课程之间有先修和后修的关系，有些课程可以任意安排次序。这种各个课程之间的次序关系可用一个称为图的数据结构来表示，如图 1-3 所示。有向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。

由以上三个例子可见，描述这类非数值计算问题的数学模型不再是数学方程，而是诸如表、树、图之类的数据结构。因此，可以说数据结构课程主要是研究非数值计算的程序设计问题中所出现的计算机操作对象以及它们之间的关系和操作的学科。

学习数据结构的目的是为了了解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。与此同时，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。

表 1-1　计算机专业课程先修关系

课程编号	课程名称	先修课程
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

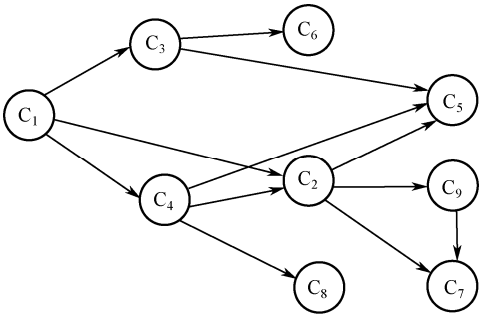

图 1-3　教学计划编排问题的数据结构

1.1.2　有关概念和术语

在系统地学习数据结构知识之前，先对一些基本概念和术语赋予确切的含义。

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，应用程序处理各种各样的数据。计算机科学中，所谓数据就是计算机加工处理的对象，它可以是数值数据，也可以是非数值数据。数值数据是一些整数、实数或复数，主要用于工程计算、科学计算和商务处理等；非数值数据包括字符、文字、图形、图像、语音等。

数据元素（Data Element）是数据的基本单位。在不同的条件下，数据元素又可称为元素、节点、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、八皇后问题中状态树的一个状态、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。

有时，一个数据元素可由若干个**数据项（Data Item）**组成，例如，学籍管理系统中学生信息表的每一个数据元素就是一个学生记录。它包括学生的学号、姓名、性别、籍贯、出生年月、成绩等数据项。这些数据项可以分为两种：一种称为初等项，如学生的性别、籍贯等，这些数据项是在数据处理时不能再分割的最小单位；另一种称为组合项，如学生的成绩，它可以再划分为数学、物理、化学等更小的项。通常，在解决实际应用问题时把每个学生记录当成一个基本单位进行处理。

数据对象（Data Object）或**数据元素类（Data Element Class）**是具有相同性质的数据元素的集合。在某个具体问题中，数据元素都具有相同的性质（元素值不一定相等），属于同一数据对象（数据元素类），数据元素是数据元素类的一个实例。例如，在交通咨询系统的交通网中，所有的顶点是一个数据元素类，顶点 A 和顶点 B 各自代表一个城市，是该数据元素类中的两个实例，其数据元素的值分别为 A 和 B。

结构（Structure）：为了理解数据结构这个概念，首先应该对“结构（Structure）”这个词汇的含义进行理解，结构这个词一般具有如下几个含义：

① 许多部件组成的事物整体；

② 事物的构造方式及组成部分的排列方式。

那么数据结构是以数据元素作为组成“部件”的一种复杂的数据组织方式。

因此，**数据结构（Data Structure）**是指数据元素及数据元素之间的关系，是复杂数据的一种组织方式。在实际问题中，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。

1.1.3　数据结构的三要素

程序或算法所处理的数据一定是有某种内在联系的。这种数据之间的内在联系就是数据之间的关系。数据之间的关系可以分成两个方面来看待，即数据的逻辑关系和数据的存储关系。

研究数据结构时，主要从 3 个方面入手，这 3 个方面称为**数据结构的三要素**，即**数据的逻辑结构**、**数据存储结构**、**数据操作**（也称为算法）。

1．数据的逻辑结构

数据的逻辑关系指的是客观上数据对象之间所具有的关系，这往往与具体的应用需求有关。比如说，一个班的学生，如果要求将所有学生按学号排列，那么学号由小到大就规定了一种逻辑关系；如果要求按拼音顺序排列，那么每个学生姓氏的拼音就规定了一种逻辑关系。

再比如，一元二次方程求根算法中，对于 3 个系数 a, b, c，尽管在算法或程序中，可以分别用 3 个独立变量表示，但它们在逻辑上仍然是有关系的，即它们同属于一个一元二次方程的 3 个系数。

数据的逻辑结构可以看成从具体问题抽象出来的数学模型，它与数据的存储无关，独立于计算机，它是从具体问题抽象出来的数学模型。

经过人们长期的研究和实践，将客观世界数据对象之间的关系归纳为以下 4 种普遍的逻辑结构。

（1）集合结构：数据元素间除“同属于一个集合”外，无其他关系。

（2）线性结构：数据可以按某种规则排列成线性表的形式，如一个班的学生名单就是一个线性表，这个班的学生之间就是一种线性关系。线性表一般表示成一个线性序列的形式：

$$(a_1, a_2, \cdots, a_i, \cdots, a_n)$$

（3）树形结构：数据之间呈现倒立的树形结构，每个元素有一个双亲，有 0 个或多个孩子。元素之间呈现一对多的关系。

（4）网状结构（图结构）：这是一种最复杂的关系，每个数据元素都可能有多个相邻的数据元素，数据元素之间呈现一种多对多的关系。

图 1-4 表示了上述 4 类基本数据结构。

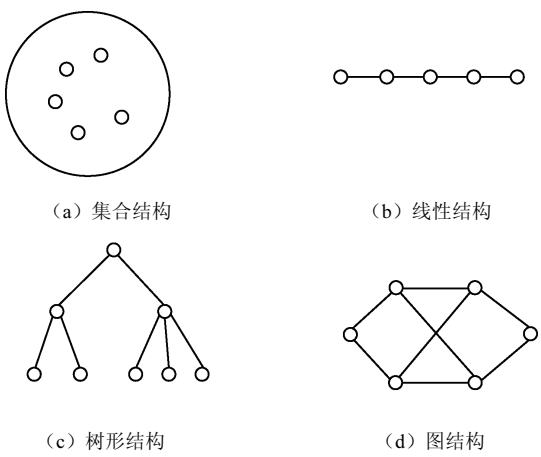

图 1-4　4 类基本数据结构的示意图

从上面所介绍的数据逻辑结构的概念中可以知道，一个数据结构（逻辑结构）有两个要素：一个要素是数据元素的集合，另一个要素是关系的集合。在形式上，数据逻辑结构通常可以采用一个二元组来表示：

Data_Structure ＝（D，R）

其中，D 是数据元素的有限集，R 是 D 上关系的有限集。

数据的逻辑结构可以看成集合论中关系理论的扩展。

2．存储结构

我们研究数据结构的目的是为了在计算机中实现对它的操作，为此还需要研究如何在计算机中表示一个数据结构。数据结构在计算机中的表示称为数据的物理结构或存储结构。存储结构所研究的是数据结构在计算机中的实现方法，包括数据结构中元素的表示及元素间关系的表示。

算法或程序所要处理的数据最终总是要在计算机存储器中存储。对于少量的数据，我们可以用单个变量表示每一个数据，每个变量在计算机中是如何存储的，我们可以不必关心。但是对于大批量的数据，为了设计一个高效的算法，必须自己定义数据的存储关系。因此除了要研究数据之间的逻辑结构外，更重要的是要研究数据的存储结构。

简单地说，数据的存储结构指的是一批数据在计算机存储器中的存储位置和存储方式，它所研究的是数据的逻辑结构在计算机中的实现方法，包括逻辑数据结构中数据元素的存储及数据元素之间关系的存储。

对存储结构的基本要求是：存储结构必须能够反映数据元素本身及数据元素之间的逻辑关系。

数据的存储结构可采用顺序存储或链式存储的方法。

目前，使用最为广泛的存储结构有以下几种：

（1）**顺序存储结构**：顺序存储方法是把逻辑上相邻的元素存储在物理位置相邻的存储单元中，由此得到的存储表示称为顺序存储结构。顺序存储结构是一种最基本的存储表示方法，顺序存储结构的典型代表是我们已经学过的数组。其基本特点是数据元素一个紧挨着一个地存放。对于逻辑上的线性表（线性结构），采用顺序存储方式时，就称为顺序表。

（2）**链式存储结构**：链式存储对数据元素在存储器中存放位置不做特殊要求，数据元素在存储器中可以随机存放。为了保持数据元素之间的逻辑关系，使用“指针”将每个数据元素联系起来。对于逻辑上的线性表，采用链式存储时，就称为链表。

除了通常采用的顺序存储方法和链式存储方法外，有时为了查找方便还采用索引存储方法和散列存储方法。

同一种逻辑结构可采用不同的存储方法，主要考虑的是运算方便及算法的时空要求。

3．作用于数据结构上的算法

作用于数据结构上的算法也称为对数据结构的操作。一般来说，不同的数据结构，其算法或操作是不一样的。但是也有几个基本的算法是大多数数据结构操作中经常遇到的，如查找、更新、插入、删除等。

总之，数据的逻辑结构与存储结构密切相关，我们在设计算法思路和框架时，更多地考虑数据的逻辑结构，而在算法实现时，则主要考虑数据的存储结构。

1.2 抽象数据类型

首先来回顾一下在程序设计语言中出现的各种数据类型。

1.2.1 数据类型

数据类型是和数据结构密切相关的一个概念。它最早出现在高级程序设计语言中，用以刻画

程序中操作对象的特性。在高级语言编写的程序中，每个变量、常量或表达式都有一个它所属的数据类型。类型规定了在程序执行期间变量的所有可能的取值范围，以及在这些值上允许进行的操作。因此，数据类型（Data Type）是一个值的集合和定义在这个值集上的一组操作的总称。

在高级程序设计语言中，数据类型可分为两类：一类是原子类型，另一类则是结构类型。原子类型的值是不可分解的。如 Java 语言中整型、字符型、浮点型、双精度型等基本类型，分别用保留字 int、char、float、double 标识。而结构类型的值是由若干成分按某种结构组成的，因此是可分解的，并且它的成分可以是非结构化的，也可以是结构化的。例如，数组的值由若干分量组成，每个分量可以是整数，也可以是另外的数组等。

1.2.2　抽象数据类型

1．抽象数据类型概念

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取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关。即不论其内部结构如何变化，只要它的数学特性不变，都不影响其外部的使用。

抽象数据类型与数据结构有着密切关系。抽象数据类型定义中的数学模型就是数据结构，定义在数学模型上的操作就是对数据结构的操作。这样我们可以将抽象数据类型的定义写成——**抽象数据类型是指一个数据结构以及定义在该结构上的一组操作**。

抽象数据类型和数据类型实质上是一个概念。例如，各种计算机都包含的整数类型就是一个抽象数据类型，尽管它们在不同处理器上的实现方法可以不同，但由于其定义的数学特性相同，在用户看来都是相同的。因此，**“抽象”的意义在于数据类型的数学抽象特性**。

但在另一方面，抽象数据类型的范畴更广，它不再局限于前述各处理器中已定义并实现的数据类型，还包括用户在设计软件系统时自己定义的数据类型。为了提高软件的重用性，在近代程序设计方法学中，要求在构成软件系统的每个相对独立的模块上，定义一组数据和作用于这些数据上的一组操作，并在模块的内部给出这些数据的表示及其操作的细节，而在模块的外部使用的只是抽象的数据及抽象的操作。这也就是面向对象的程序设计方法。

抽象数据类型的特征是使用与实现相分离，实行封装和信息隐蔽。就是说，在抽象数据类型设计时，将类型的定义与其实现分离开来。

抽象数据类型具有以下特点：

（1）抽象数据类型由基本的数据类型组成。

（2）抽象数据类型反映了数据的逻辑结构及其在逻辑结构上定义的操作。

（3）抽象数据类型中的操作是一种抽象操作，独立于具体实现。抽象数据类型本身将数据和操作封装以实现信息隐蔽。

2．抽象数据类型的描述方法

抽象数据类型可用三元组（D，S，P）表示，其中，D 是数据对象，S 是 D 上的关系集，P 是对 D 的基本操作集。抽象数据类型的一般描述模式如下：

```
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〉
}
```

【例 1-4】抽象数据类型复数的定义。

```
ADT Complex
{
   数据对象：D＝{e1,e2 | e1,e2∈RealSet  }
   数据关系：R1＝{<e1,e2> | e1 是复数的实数部分| e2 是复数的虚数部分
   基本操作：
     InitComplex( &Z, v1, v2 )
       操作结果：构造复数 Z,其实部和虚部分别被赋予参数 v1 和 v2 的值
     DestroyComplex( &Z)
       操作结果：复数 Z 被销毁
     GetReal( Z, &RealPart )
       初始条件：复数已存在
       操作结果：用 realPart 返回复数 Z 的实部值
     GetImag( Z, &ImagPart )
       初始条件：复数已存在
       操作结果：用 ImagPart 返回复数 Z 的虚部值
     Add( z1,z2, &sum )
       初始条件：z1,z2 是复数
       操作结果：用 sum 返回两个复数 z1,z2 的和值
} ADT Complex
```

在程序设计过程中，要使用具体程序设计语言提供的有关方式进行描述。在 Java 语言中我们直接使用 Java 接口描述抽象数据类型。

【例 1-5】 ADT Complex 的 Java 接口描述。

```
public interface Complex
{
    //数据对象：double real, imag;
    //基本操作
    public  Complex copy（Complex  z）;
    public  double GetReal（）;
    public  double GetImag（）;
    public  Complex add（Complex  z1，complex  z2）;
}
```

1.3 算法概念及算法设计的问题

算法与数据结构的关系紧密，在算法设计时先要确定相应的数据结构，而在讨论某一种数据结构时也必然会涉及相应的算法。

1.3.1 什么是算法

公元 825 年，一位名叫阿尔·花剌子模（Al - Khwarizmi）的波斯数学家编写了一本教

科书，书中概括了进行数字四则算术运算的法则。现代名词“算法”（Algorithm）就来源于这位数学家的名字。后来，美国 Webster’s 字典中将其定义为求解某种问题的任何专门的方法。这里只是给出在计算机科学里中的一种解释，不做精确定义。

算法是用计算机解题的精确描述，算法就是逐步（Step-by-Step）执行某类计算的方法，一个算法描述的是有穷的动作的序列或步骤。

【例 1-6】（求三个数的最大值）设计一个算法对任意给定的三个整数 x, y, z，求出其最大值。这是一个简单的算法，只要先比较出 x 和 y，得到较大的值，再用这个值与 z 比较，将两者中较大的值作为结果输出即可，可将这个算法描述如下：

① 输入变量 x, y, z 的值；

② 比较 x 和 y，如果 x>y，则 max=x，否则 max=y；

③ 比较 max 与 z，如果 max<z，则 max=z；

④ 输出结果 max。

对一个要求解的问题，算法就是解决问题的思想和方法。那么程序是什么呢？程序是算法的具体实现，没有算法，程序就成了无本之木，无源之水。有了算法，将它表示成程序就不困难了。

虽然“算法”一词在公元 825 年左右出现，但算法在这之前就早已出现，可以说是源远流长。而计算机的出现，开创了算法研究的新时代。

一个古老的非常著名的算法是用于求两个整数的最大公约数的欧几里得算法。这个算法最早出现在公元前 350～公元前 300 年由欧几里得写成的 *Elements*（几何原本）中。算法的发明人是欧几里得，因此，人们自然将该算法称为欧几里得算法，欧几里得算法也称为辗转相除法。具体思路如下：

已知两个整数 x 和 y，用 mod（x,y）或者 x mod y 表示 x 被 y 除后所得的余数。两个已知数 a 和 b（假设 a>b）的最大公约数的计算过程如下:

设 r_1=mod（a,b），再设 r_2=mod（b,r_1），r_3=mod（r_1,r_2），一直计算下去，直到 mod（r_{n-1},r_n）=0，这时 r_n 就是 a 和 b 的最大公约数。

【例 1-7】计算 91 和 52 的最大公约数，求解过程如下。

① mod(91,52) =39

② mod(52,39) =13

③ mod(39,13) =0

所以，13 就是 91 和 52 的最大公约数。

自然语言表示的欧几里得算法如下。

算法 1-1　欧几里得算法求最大公约数

输入：两个正整数 m 和 n

输出：m 与 n 的最大公约数(公因子)

步骤 1:求余数，以 n 除 m 并令 r 为所得余数(0≤r＜n)

步骤 2:余数 r 为 0 吗？若 r=0，算法结束；n 即为答案

步骤 3:互换,置 m=n,n=r，转步骤 1

1.3.2 算法特性

算法（Algorithm）是对特定问题求解步骤的一种描述，是指令的有限序列。其中每一条指令表示一个或多个操作。一个算法应该具有下列特性：

（1）有穷性。一个算法必须在有穷步之后结束，即必须在有限时间内完成。

（2）确定性。算法的每一步必须有确切的定义，无二义性。算法的执行对应着的相同的输入仅有唯一的一条路经。

（3）可行性。算法中的每一步都可以通过基本运算的有限次执行得以实现。

（4）输入。一个算法具有零个或多个输入，这些输入取自特定的数据对象集合。

（5）输出。一个算法具有一个或多个输出，这些输出同输入之间存在某种特定的关系。

算法的含义与程序十分相似，但又有区别。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，它将永远不会停止，即使没有作业需要处理，它仍处于动态等待中。因此，操作系统不是一个算法。另一方面，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。算法代表了对问题的解，而程序则是算法在计算机上的特定的实现。一个算法若用程序设计语言来描述，则它就是一个程序。

算法与数据结构是相辅相成的。解决某一特定类型问题的算法可以选定不同的数据结构，而且选择恰当与否直接影响算法的效率。反之，一种数据结构的优劣由各种算法的执行来体现。

1.3.3 算法的结构和表示方法

算法是执行任务的步骤或指令序列，步骤或指令是有先后次序的。这种指令的执行次序称为**执行流程**。对不同的问题，其执行流程可能不同，因此就存在流程控制问题，在算法设计中，将流程控制简称为**算法结构**。经过多年的总结和研究，人们发现无论多么复杂的算法，都可以用下面三种流程控制结构来描述：

- 顺序结构
- 选择结构
- 循环（重复结构）

如何用这些结构表示一个算法呢？这就是算法的表示方法，或者称为算法的描述方法。设计好一个算法之后，不能只是自己心知肚明，必须准确、清楚地将所设计的解题步骤记录下来，或提供交流，或编写成程序供计算机执行。

常用的算法描述方式有：程序流程图、程序设计语言、自然语言与伪代码等。

1．用流程图来表示算法

流程图语言是人们经常用来描述算法的工具。流程图用规定式样的图形、指向线和文字说明组合起来表示算法。其优点是直观、清晰、易懂，便于交流；缺点是不便于修改，不便于表示成计算机文档。

2．用程序设计语言表示算法

用计算机程序设计语言表示算法一步到位，写出的算法能由计算机处理。事实上，由程序设计语言描述的算法就是计算机程序（一般表现为一个子程序或程序片段）。优点是程序

设计语言表示的算法可直接编译执行，对于懂得这种语言的人而言，通过运行程序可以马上验证算法的正确性。缺点也很明显，其一，不便于体现自顶向下、逐步求精的思想；其二，程序设计语言有很多细节性的新的要求，因此在进行算法设计时，算法主要思想往往会被一些细节性的东西所淹没。

3．用伪代码表示算法

所谓伪代码，实际上是一种主要让人来看的程序，它结合了自然语言表示和程序设计语言的优点，丢弃了程序设计语言中的烦琐细节，保留了程序设计语言中的关键的流程控制结构，再适当辅之以自然语言描述。

由于伪代码借鉴了程序设计语言中的主要控制结构，同时又允许使用自然语言，因此是一种非常好的算法描述方式，而且也很容易转换成具体的程序。本书在后面的内容讲解中主要使用伪代码描述算法。

【例 1-8】（闰年的判定）根据有关的研究结果，判断闰年的条件（给定一个年份 k）可以表示为：

（1）能被 4 整除，但是不能被 100 整除的年份是闰年；

（2）能同时被 100 和 400 整除的年份是闰年。

用伪代码表示的算法如下。

算法 1-2　闰年判定算法的伪代码描述

```
第 1 种描述：
input(k) ) //输入 k，k 表示当前年份
if ((k mod 4 = 0) and (k mod 100 ≠ 0)) or ((k mod 100 = 0)
and (k mod 400 = 0)) then
   output(“是闰年”)
else
   output(”不是闰年”)
end if
==========================================================================
第 2 种描述：
输入（k）//k 表示当前年份
if （k 能被 4 整除 and k 不能被 100 整除） or （k 能被 100 整除 and k 能被 400 整除） then
   输出(“是闰年”)
else
   输出(“不是闰年”)
end if
```

上面给出了同一种算法的两种伪代码描述：第一种描述更接近程序设计语言描述，第二种描述更接近自然语言描述。可以看出，伪代码描述算法非常灵活，自然语言的使用可多可少，以便于其他人能够读懂并且易于转换为具体程序为原则。事实上对于稍有程序设计经验的人来说，上面两种算法都很容易转换成具体的程序。

1.3.4　算法设计原则

算法设计是程序设计的基础，算法设计也应遵循程序设计的基本思想和原则。其中结构

化程序设计思想具有重要的影响，结构化程序设计的一个重要原则是“自顶向下，逐步求精”。

人类大脑的思维能力是有限的，在同一时刻，人不可能记忆、思考太多的问题。面对复杂的问题，我们很难同时把所有的细节都想清楚，只能首先把大的框架搞清楚，然后再分析解决细节问题。

在算法和程序设计中，我们一般遵循下面的步骤进行：

① 明确算法的输入、输出数据

在设计算法和程序之前，首先应该弄清楚输入、输出数据是什么。这一步实际上可以更好地帮助我们理解题意。

② 自顶向下，逐步求精

从总体出发，“居高临下”，逐层分解和逐步细化。首先抛开细节设计出抽象算法和数据结构，然后把抽象数据和操作逐步具体化，直到可以由计算机具体实现为止。

③ 模块结构

在程序设计中，将一个复杂的算法（或程序）分解成若干个相对独立、功能单一的模块或者函数，利用这些模块或函数即可有效地组合成所需要的全局算法（或）程序。

1.3.5 几种基本的算法设计方法和策略

算法设计是一个创造性的过程，人们在长期的算法研究中创造了许多设计算法的方法和策略，使得算法设计有规可循，有据可依。大致说来，算法设计主要有以下的一些策略和方法和策略：

（1）穷举策略；
（2）递推与递归策略；
（3）分而治之策略；
（4）回溯策略；
（5）贪心策略；
（6）动态规划策略。

对于这些不同的算法设计策略，在后续的章节中，将分别介绍。

1.3.6 编程解决问题的一般步骤

1. 分析问题

分析问题的步骤如下：

① 分析问题给定的条件、要求，分析的结果是将条件、要求表示成数据的形式；
② 分析解决问题的思路，结果是形成算法设计的原则和策略；
③ 分析输入、输出。

2. 概要设计

概要设计的步骤如下：

① 根据上一步的分析结果，规划数据的组织方式。

② 将算法设计思路、原则、策略写成概要性的算法框架。

3．详细设计

详细设计是将算法框架中的一些关键算法细化，直到可以用具体的程序设计语言（如 C 语言）表达为止。

4．算法实现、调试与测试

算法实现、调试与测试步骤如下：

① 将伪代码或者框图描述翻译成 C 语言程序；

② 上机调试程序，查找错误；

③ 设计输入数据，对程序进行测试。

5．整理、编写文档

根据需要，将前面几个步骤所得到的结果整理成文字形式保存起来，以备今后查阅、使用。

1.4　算法分析

在学习数据结构时，总是要学习与数据结构相关的一些重要算法。正如在日常生活中完成某一项任务往往有不同的办法一样，用计算机解决一个问题也往往有多种不同的算法。虽然这些不同的算法都能达到解决同一个问题的目的，但是它们的执行过程和执行效率可能差别很大，为此必须对算法的基本性能进行分析和评价。

通常从正确性、可读性、健壮性、效率等几个方面来评价算法的性能。

正确性 算法的执行结果应当满足预先规定的功能和性能要求。

可读性 一个算法应当思路清晰、层次分明、简单明了、易读易懂。

健壮性 当输入不合法数据时，应能作适当处理，不至于引起严重后果。

效率 算法的效率指的是时间效率和空间效率。所谓时间效率是指执行一个算法所花费的计算机时间；而空间效率指的是执行一个算法说需要的内存空间的大小。

尽管评价一个算法性能的要素有不少，但是从理论分析的角度看，一般讲“算法分析”更多地指的是**算法的时间效率分析**，算法的时间效率分析也称为**算法时间复杂性分析**。

所谓算法的复杂性指的是在计算机上运行算法所需要的资源量。对于计算机来说，最重要的资源是时间（CPU 时间）和空间（内存空间）。因此一般意义上讲的算法分析主要是指算法的**时间复杂度**分析和**空间复杂度**分析。

什么是算法的时间复杂度呢？不十分严格地讲，算法的时间复杂度就是执行一个算法所需要的“时间”的长短，这里的时间指的是占用机器的 CPU 时间；同样道理，空间复杂度是指执行一个算法所需要的存储空间的大小。

由于空间复杂度分析和时间复杂度分的方法基本相同，所以下面主要介绍时间复杂度分析。

1.4.1　时间复杂度分析的两类方法

时间复杂度分析的方法可以分为以下两类。

（1）理论分析方法

基本思想是在一定的简化假设下，找出算法的运行“时间”与问题规模 n 的函数关系，这样，只要知道问题的规模 n，就可以直接估算出算法的执行“时间”。进一步来说，就是根据时间 T 与 n 的具体函数关系，分析当规模 n 很大时，函数 T(n)的渐进变化率（变化速度），这称为渐进阶分析。根据渐进阶的类型，将算法分成不同的类型。

（2）实验分析方法

在下述几种情况下需要进行算法时间复杂度的实验分析。

第一种情况，许多算法非常复杂，理论分析不可行，这个时候需要进行算法时间复杂度的实验分析。

第二种情况，许多算法本质上都是指数时间复杂度，属于 NP-Hard 问题范畴。尤其是人工智能领域的算法，基本上都是指数时间复杂度，继续进行理论分析已经没有意义。对这些指数时间复杂度算法，当数据量比较大的时候，即使一点小的改进，算法的运行时间也会大大减少。对于这一类算法，只有通过实验分析才能比较各种算法的优缺点。

第三种情况，在大数据时代，处理的数据都是海量数据。对于海量数据，算法采取一点小的改进，有时也有非常重要的意义。这时，理论分析方法已经不起作用。

理论分析方法是一种“粗放型”的分析方法。而实验分析可以看成“精细型”的分析方法。实验分析是计算机及相关学科算法研究的基本方法。

1.4.2 时间复杂度分析的理论框架

一个程序的时间复杂度（Time Complexity）是指程序运行从开始到结束所需要的“时间”。那么一个算法 A 在计算机 M 上运行时，有哪些因素会影响算法的运行时间呢？简单思考一下，可以归纳出下面一些因素。

（1）机器的硬件速度（M）。不同的机器，运行同一个算法，使用的机器时间是不一样的。

（2）书写程序的语言。同一个算法用不同的语言实现，其执行时间一般是不一样的。一般情况下，解释型语言的程序运行速度慢，编译型语言的程序运行速度快。解释器或者编译器本身的质量也同样影响算法的运行时间。

（3）数据的规模（n）。算法处理的数据规模越大，执行算法所需时间就越长。例如，求 1000 以内的素数与求 100 以内的素数，其运行时间必然是不同的。

（4）数据的分布（D）。原始数据的排列方式也会影响算法的执行时间。

在以上因素的影响下，对算法时间复杂度进行理论分析是不现实的。科学研究的一种基本手段是抽象，即在一定的假设下，舍弃那些不重要的因素，只考虑最本质的影响因素。

对于这里的问题，应该做出什么样的假设，舍弃哪些非主要因素呢？

假设 1：所有的算法都在同一台机器上运行和比较。

在这个假设下，我们可以将影响因素（1）去掉，任何算法都在同一台机器上运行的话，CPU 的运算速度对算法的影响可以排除。

假设 2：所有的算法都使用同一种语言编写，使用同样的编程风格。程序语言对算法实现已经达到最优化。

在这个假设下，影响因素（2）也可以去掉。

在上述两个假设下，去掉一些次要的影响因素后，真正影响算法运行速度的因素只有数据规模和数据的初始分布。假设算法的运行时间为 T，那么可表示成下面的形式：

$$T = f(D, n)$$

这是一个非常重要的公式，尽管这只是一个抽象的函数关系，但是后续理论分析和实验分析都是基于该函数关系。

在上述时间复杂度的抽象公式中，需要解决下面几个问题：

- 时间 T 的度量单位；
- 数据规模 n 的确定；
- 如何处理输入数据的初始分布 D；
- 复杂的函数 f 如何简化，便于将时间复杂度分类。

1．数据规模 n 的确定

数据规模指的是数据量的大小，一般用 n 表示。几乎所有的算法，处理的数据量越大，所需的运行时间就越长。例如，100 个数据的排序比 10 个数据的排序时间要长，这里数据规模分别是 100 和 10。同样道理，2 个 10 阶矩阵的相乘比 2 个 3 阶矩阵相乘所花费的时间要长，这里数据规模分别是 10 和 3。求 1000 以内的素数显然比求 100 以内的素数花费的时间长，这里数据规模分别是 1000 和 100。

一般情况下，输入数据的个数就是其规模 n。但是在某些时候，输入数据的个数并不就是数据规模 n。例如，输入一个整数 n，将整数 n 的每一位数字分解出一个算法，其数据规模是整数 n 的数字个数。

2．确定基本操作

时间复杂度公式中的 T 使用什么时间单位呢？使用分？秒？或者小时？看起来都不合适。这样的时间单位称为**绝对时间单位**，使用绝对时间单位不便于算法时间的比较，同时绝对时间单位也无法揭示算法的本质。那么算法的本质是什么呢？

算法的本质是，无论多么复杂的算法，都可以分解成一些简单的基本操作或运算，换句话说，一些基本运算在流程控制结构的控制下反复执行构成了算法。看下面一段算法代码：

```
s=0;
for(i=1;i<=n;i++)
    s=s+i;
```

这段代码中，i 是循环变量，n 是数据的规模。算法可以抽象出 2 个基本运算：赋值运算和加法运算。假设一次赋值运算所需要的时间是 c_1 秒，一次加法运算的时间是 c_2 秒。不考虑循环控制所花费的时间，那么本算法总的运行时间为

$$T=c_1+c_2*n+c_2*n$$

再进一步假设，c_1 与 c_2 相差不大，即 $c_1 \approx c_2$ 时，上式两边同除以 c_1，则：

$$\frac{T}{c_1} = 1 + 2n$$

T 本身是绝对时间，那么 $\frac{T}{c_1}$ 就是**相对时间**，其含义是基本操作执行的次数。为了不引入过多的符号，今后还是用 T 表示基本操作的执行次数（相对时间）。即

$$T=1+2n$$

这样，在分析算法时，只需要考虑基本操作的执行次数与数据规模之间的关系即可。

简单总结一下，一个算法是由控制结构和原操作构成的，其执行时间取决于两者的综合效果。为了便于比较同一问题的不同算法的实现效果，通常的做法是：从算法中选取一种对于所研究的问题来说是基本运算的操作，以该操作重复执行的次数作为算法的时间度量。

那么什么是基本操作呢？所谓**基本操作**，指的是在算法运行时间度量中，占主导优势的操作。不同的问题，其基本操作都有所不同。表 1-2 为几种算法的基本操作及数据规模。

表 1-2　输入规模及基本操作示例

问　　题	基本操作	数据规模
在有 n 个元素的表中搜索关键字	关键字比较	表中元素个数 n
两个浮点数矩阵相乘	浮点数相乘	矩阵维数
计算 a^n	浮点数相乘	N
图问题	顶点或边的遍历	顶点数或边数

3．处理数据分布

根据公式 T=f（D, n），影响 T 的因素中，数据分布 D 是非常复杂的，即使我们能给出数据的概率分布函数，数学处理依然很复杂，因此必须想办法对数据分布 D 进行特殊处理，以简化时间复杂度分析。首先看一个例子。

假如我们要对如下的数据集合 X 使用插入排序算法升序排序

X=｛32，12，44，54，21，7，13，25，30｝

考虑数据的以下三种分布：

第一种情况：集合 X 中的数据初始分布是升序有序的，用 D1 表示

D1=（7, 12, 13, 21, 25, 32, 30, 44, 54）

第二种情况：集合 X 中的数据初始分布是降序有序的，用 D2 表示

D2=（54, 44, 32, 30, 25, 21, 13, 12, 7）

第三种情况：集合 X 中的数据初始分布是随机的，用 D3 表示

D3=（32, 12, 44, 54, 21, 7, 13, 25, 30）

显然，在这三种数据分布情况下，采用插入排序算法对这一批数据进行升序排序所需要的时间是不一样的。在分布为 D1 的情况下，所需时间最短，这是最好的一种情况。在第二种情况下，排序所需时间最长，这是最坏的一种情况。在第三种情况下，每个数据元素的出现位置是随机的，可以用概率分布描述，比如最简单的均匀分布，在已知概率分布的情况下，可以求出排序的平均时间。因此这种情况也称为“平均情况”。

概括一下，对于数据分布，我们用三种极端情况来代替复杂的概率分布。这三种情况是最好情况、最坏情况以及平均情况。即

$$T_{worst}=f_w(n)$$

$$T_{best}=f_b(n)$$

$$T_{avg}=f_a(n)$$

也就是说，通过分析三种情况下的时间复杂度来代替数据的分布。

那么是否对每一种算法都要分析最好情况、最坏情况和平均情况下的时间复杂度呢？答案是否定的。大多数情况下，只需要分析最坏情况下的时间复杂度。因为如果一个算法的最坏情况时间复杂度能被人们接受，那么最好情况和平均情况时间复杂度当然也能被接受。另一方面，某些算法的最好、最坏、平均情况的时间复杂度是相同的。在后续的讲解中，我们用 T 来代替 T_{worst}，即一般情况下的 T=f (n)表示的是最坏情况下的时间复杂度关系式。

4.（函数关系）的简化与渐进分析

找到算法执行时间 T（基本操作执行次数）与数据规模之间的具体关系是整个时间复杂度分析中最重要也是最困难的一步。

函数 T(n)往往是复杂的数学公式。不同的算法，其数学表达式 T(n)是不同的。这对于算法时间复杂度比较来说，是不够直观和方便的。于是数学家们发明了一种新的表示方法，**函数的界**或者称为**函数的阶**。什么意思呢？ 就是考虑当 n 足够大时，将函数 T(n)分成几种类别。任何一种算法，其时间复杂度一定属于某种类别。这一类分析称为**时间复杂度的渐进分析**。

考虑以下两个算法。

算法 1

```
x=-0;
for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
      x=i*j;
```

数据规模是 n，基本操作是 x=i*j，时间复杂度的函数关系式为

$$T_1=\sum_{i}^{n}\sum_{j}^{n}1=n^2$$

算法 2

```
x=-0;
for(i=1;i<=n;i++)
   for(j=1;j<=i;j++)
      x=i*j;
```

数据规模是 n，基本操作是 x=i*j，时间复杂度的函数关系式为

$$T_2=\sum_{i}^{n}\sum_{j}^{n}1=\frac{1}{2}n^2+\frac{1}{2}n$$

时间复杂度函数 $T_1=n^2$ 与 $T_2=\frac{1}{2}n^2+\frac{1}{2}n$ 在渐进意义上是相同的。因为当 $n\to\infty$ 时，T_1 与 T_2 是同阶的，事实上，有

$$\lim_{n\to\infty}\left(\frac{T_1}{n^2}\right)=1$$

$$\lim_{n\to\infty}\left(\frac{T_2}{n^2}\right)=\frac{1}{2}$$

也就是说，当$n \to \infty$时，T_1与T_2都与n^2同阶，相差的仅仅是常数。这样我们将T_1与T_2表示成下面的形式

$$T_1 = O(n^2), \quad T_2 = O(n^2)$$

这种表示法称为“大O”表示法。“大O”表示法的本质是，只取函数变化最快的项代表时间复杂度的阶（也称为时间复杂度类型）。该记号是保罗·巴克曼（P.Bachmann）于1892年在《解析数论》（*Analytische Zahlentheorie*）一书引入的，它为Order（数量级）的第一个字母，它允许使用“=”代替“≈”，如$n^2+n+1=O(n^2)$。该表达式表示，当n足够大时表达式左边约等于n^2。

设f(n)是一个关于正整数n的函数，若存在一个正整数n_0和一个常数C，当$n \geqslant n_0$时，$|T(n) \leqslant C|f(n)|$均成立，则称f(n)为T(n)的同数量级的函数。于是，算法时间复杂度T(n)可表示为：

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

常见的表示形式有常数级O(1)、对数级$O(\log_2 n)$、线性级O(n)、多项式级$O(n^c)$，指数级$O(c^n)$和阶乘级O(n!)等，表1-3所示为典型的时间复杂度类型。图1-5所示为典型的时间复杂度类型随数据规模n的变化情况。（本书中$\log_2(n)$简写成logn）

表1-3　常用的时间复杂性渐进类

效率类型	含　义
O（1）	常数阶（无重复执行）
O（logn）	对数阶
O（n）	线性阶
O（logn）	对数线性阶
O（n^2）	平方阶
O（n^3）	立方阶
O（2^n）	指数阶
O（n!）	阶乘阶

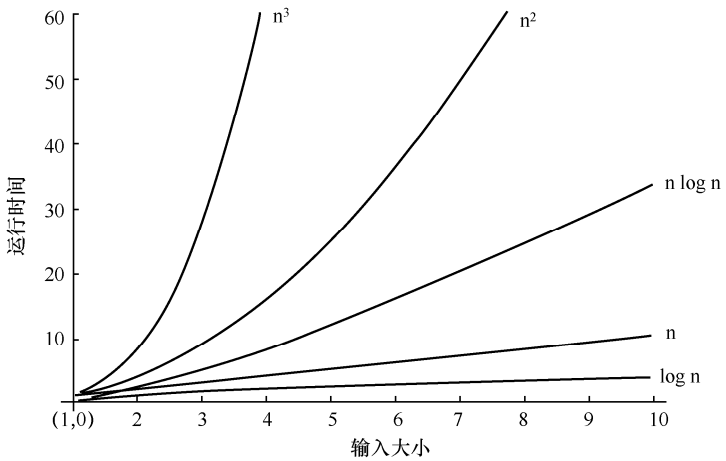

图1-5　一些表示运行时间的典型函数增长情况