第1章 概 论

算法与数据结构是计算机科学与技术、软件开发与应用、信息管理、电子商务、网络安全等相关专业的一门专业基础课，它不仅是计算机学科的理论基础之一，也是计算机系统软件和应用软件开发者的必备基础，读者无论是从事计算机行业，还是希望在计算机方面继续深造，该课程的学习都是必须的。

算法与数据结构理论发展至今，已成为一门比较成熟的课程。它的应用范围已经渗透到编译系统、操作系统、数据库、人工智能、信息科学、企业管理、系统工程、应用数学、计算机辅助设计及其他信息管理的应用中。

算法与数据结构专门研究从解决非数值计算的现实问题中抽象出来的数据在计算机中如何表示、快速存取和处理的方法。这里所说的数据是广义的概念，它不仅包括数值数据、字符数据、逻辑数据等简单数据，而且还包括带有一定结构的各种复杂的数据，如字符串、记录、向量、矩阵等，也包括各种表格、图形、音频和视频。

当用计算机存储数据时不仅要存储这些数据的值，而且相应地还要存储这些数据之间的相互关系。如何存储数据和这些数据之间的关系就出现了各种不同的存储方法。

1.1 什么是数据结构

计算机的应用可以分为科学计算和生产过程自动控制、管理以及数据处理。一般来说，用计算机解决一个实际问题时，需要经过以下几个步骤：首先从具体问题抽象出一个适当的数学模型，其次选择或设计一个解此数学模型的算法，最后编写程序进行调试、运行，直至得到最终的解答。在此过程中寻求数学模型的实质是分析问题，从中提取操作对象，并找出这些操作对象之间的关系，然后用数学语言加以描述。例如，求解建筑结构工程中的结构静力的分析计算，首先要利用有限元的分析方法得到一个线性代数方程组的数学模型；利用计算机进行全球天气预报需要求解一组球面坐标系下的一般环流模式方程，这个复杂的方程就是天气预报问题的数学模型。这些问题中涉及大量的数值计算，它们的数学模型可以用微分方程、常微分方程、多元函数微分方程、线性微分方程、代数方程、积分方程等表示。但是多数非数值计算问题无法用数学方程进行描述。请看如下3个例子。

例1.1 考生录取信息系统。

若某高校需要在报考该学校的考生中查找某个考生的有关情况，或者想查询报考某个专业的考生的有关情况，或者统计某个分数段的考生的情况，则可以建立相关的数据结构，并且按照某种算法编写相关程序，这样就可以实现计算机自动检索。因此，可以在考生录取信息系统中建立一张按考号顺序排列的考生信息表和分别按姓名、专业、成绩顺序排列的索引表，如图1-1所示。由这四张表构成的文件便是学生信息检索的数学模型，计算机的主要操作有按照某个特定要求（如给定考号）对考生信息文件进行查询。

诸如此类的表结构还有学生学籍管理系统、电话自动查号系统、图书馆的书目管理系统、仓库管理系统和人事档案管理系统等。在这类问题中，计算机处理的对象是各种表，元素之间存在着一种简单的线性关系，施加于对象上的操作有查询、插入和删除等，因此这类问题的数学模型就是各种表格，而插入和删除等操作都是以查找为基础，所以查找算法是解决这

类问题的主要算法。

考号	姓名	性别	报考专业	成绩
2300411	李闽志	男	计算机科学与技术	658
1000472	于惠芳	女	英语	632
1506302	刘　红	女	应用数学	617
2105902	宋大明	男	英语	600
0934785	高大庆	男	计算机科学与技术	601
0600807	何文丽	女	英语	611
0878529	隋文涛	男	应用数学	612
1690834	崔秀海	男	英语	602
1710641	于众群	女	计算机科学与技术	619

a) 考生信息表

崔秀海	8
高大庆	5
何文丽	6
李闽志	1
刘　红	3
宋大明	4
隋文涛	7
于众群	9
于惠芳	2

b) 姓名索引表

计算机科学与技术	1, 5, 9
英语	2, 4, 6, 8
应用数学	3, 7

c) 专业索引表

600~609	4, 5, 8
610~619	3, 6, 7, 9
620~629	
630以上	1, 2

d) 成绩索引表

图1-1　考生信息系统中的数据结构

例1.2　人 - 机博弈。

计算机之所以能和人博弈是因为已经将对弈的策略输入到了计算机中。由于对弈的过程是在一定的规则下随机进行的，因此，为使计算机能够灵活对弈，就必须把对弈过程中所有可能发生的情况以及相应的对策都加以考虑，并且，一个“好”的棋手在对弈时不仅要看棋盘的状态，还要预测棋局发展的趋势，直至最后结局。所以在对弈问题中，计算机操作的对象是对弈过程中可能出现的称之为格局的棋盘状态。图1-2所示的井字棋对弈树，包括了多个对弈的格局，格局之间的关系是由比赛规则决定的。通常，这个关系是非线性的，因为从一个棋盘格局可以派生出几个格局，而从每一个新的格局又可派生出多个可能的格局。因此，如果将从对弈开始到结束的过程中所有可能出现的格局都表示出来，就可以得到一棵“树”。其“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每个叶子的过程。

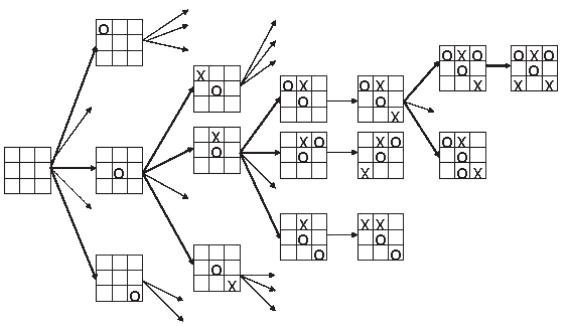


图1-2　井字棋对弈树

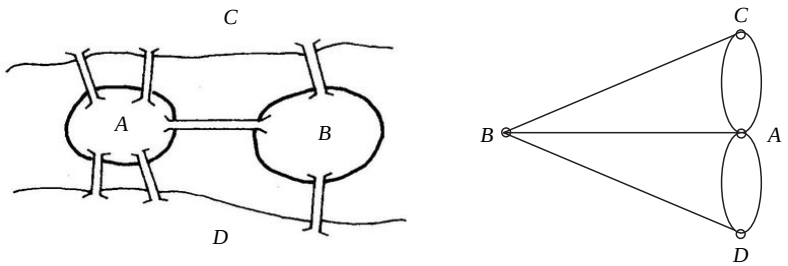
诸如此类的树结构还有家族的族谱、计算机文件系统以及一个单位的组织机构等。在这类问题中，计算机处理的对象是树形结构，元素间的关系是一种层次关系，施加于对象上的操作有查询、插入和删除等，此类问题的数学模型就是如何表示棋盘和棋子，算法就是博弈的规则和策略。

例1.3　哥尼斯堡七桥问题。

七桥问题十分有趣，对于图论这一学科的建立也有很重要的意义，它提供了一个很好地把现实生活模型抽象为图问题的实例。问题的背景如下：在18世纪东普鲁士的哥尼斯堡有许多人热衷于这样一个游戏：由于哥尼斯堡被普莱格尔河分成四块，它们之间通过七座桥互相连接，如图1-3a所示，人们想知道，怎样才能够从某块陆地出发，经过每座桥一次且仅一次最后回到出发点。这一问题一直没有人找到答案。1736大数学家欧拉把两个小岛和南北两岸抽象为四个点A、B、C、D，而把这些桥抽象为连接两个点的一条线，这样，哥尼斯堡七桥问题的图表示如图1-3b所示。欧拉证明，如果存在经过每条边一次且仅一次回到出发点的路径，则充分必要条件是：

1) 图是连通的。

2) 在图中与每个顶点相连的边数必须是偶数。



a) 哥尼斯堡七桥　　b) 哥尼斯堡七桥问题的图表示

图1-3　哥尼斯堡七桥问题

由于哥尼斯堡七桥问题不满足这样的条件，所以它是无解的。欧拉由七桥问题所引发的对图的研究论文是图论的开篇之作，因此欧拉也被称为图论之父。从七桥问题可以看到，要利用图来解决问题，关键的一步是找到现实问题的实体与图的点和边的对应关系。

诸如此类的结构还有城市网络交通图和网络工程图等。这类问题中计算机处理的对象是各种图，元素间的关系是复杂的图形或网状关系，是一种多对多的关系，施加于对象上的操作依然有查询、插入和删除等，此类问题的数学模型就是图或网络，算法是如何求两点之间的距离或最短路径等。

上述三个例子表明，描述这类非数值计算问题的数学模型不再是数学方程式，而是诸如表、树和图之类的数据结构。因此，简单地说，数据结构是一门研究非数值计算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的学科，用以描述现实世界实体的数学模型（非数值计算）及其上的操作在计算机中的表示和实现。

数据结构在计算机科学中是一门综合性的专业基础课。数据结构的研究不仅涉及计算机硬件（特别是编码理论、存储装置和存取方法等）的研究范围，而且和计算机软件的研究有着密切的关系，无论是编译程序还是操作系统都涉及数据元素在存储器中的分配问题。在研究信息检索时也必须考虑如何组织数据，以使查找和存取数据元素更为方便。因此，可以认为数据结构是介于数学、计算机硬件和软件三者之间的一门核心课程。

1.2　数据结构的基本概念和术语

在系统地学习算法与数据结构知识之前，先对相关的概念和术语赋予确切的含义。

数据（Data）是信息的载体，是描述客观事物的数、字符，以及所有能输入到计算机中并被计算机程序识别和处理的符号的集合。数据是计算机处理的信息的某种特定的符号表示

形式，如数学计算中所用到的整数和实数，文本编辑所用到的字符串等都是数据。随着计算机软、硬件技术的发展，计算机能够处理的对象也在扩大，相应的数据的含义也被拓宽了。文字、图像、图形、声音和视频等非数值数据也都是计算机可以处理的数据。

数据元素（Data Element）是数据中的一个“个体”，是数据的基本单位。在有些情况下，数据元素也称为元素、结点、顶点、记录等。数据元素用于完整地描述一个对象，如一个考生记录、树中棋盘的一个格局（状态）、图中的一个顶点等。

数据项（Data Item）是组成数据元素的、有特定意义的、不可分割的最小单位。如构成一个数据元素的字段（域、属性等）可称之为数据项。又如考生信息表中的学号、姓名、性别、成绩等。

数据元素是数据项的集合。

数据对象（Data Object）是性质相同的数据元素的集合，是数据的一个子集。如整数数据对象是集合$N = \{ 0, \pm1, \pm2, \cdots \}$，字母字符数据对象是字符集合$C = \{$‘a’，‘b’，…，‘z’$\}$，学生数据对象等。

数据结构（Data Structure）是指相互之间存在着一种或多种特定关系的数据元素的集合。简言之是带结构（Structure）的数据元素的集合。所谓结构就是元素之间存在的一种或多种特定的约束关系。它是根据人们解决实际问题的需要和问题本身所含数据之间的内在联系而抽象出来的。这种数据结构与如何利用计算机存储和处理无关，所以一般被称为数据的逻辑结构。根据数据元素间关系的不同特性，一般有下列四种基本类型的逻辑结构：

1) 集合结构　集合结构中的数据元素除了仅仅同属于同一个集合外，不存在逻辑关系。

2) 线性结构　线性结构中的数据元素之间存在着一种一对一的关系。这种结构的特征是：若结构是非空集，则有且仅有一个开始结点和一个终端结点，并且所有结点最多只能有一个（直接）前驱和一个（直接）后继。

3) 树形结构　树形结构中的数据元素之间存在着一种一对多的关系。在这种结构中，除了一个特殊的结点（称之为根结点）外，其他所有结点都有且仅有一个前驱结点和零至多个后继结点。

4) 图形结构　图形结构中的数据元素之间存在着一种多对多的关系。在这种结构中，所有结点均可以有多个前驱和多个后继。图形结构也称为网状结构。

数据结构的形式化定义为：数据结构是一个二元组

$$Data\_Structure = (D, R)$$

其中，D是数据元素的有限集合，R是D上关系的有限集合，这里每个关系都是从D到D的关系。在表示每个关系时，用尖括号表示有向关系，如〈a，b〉表示存在结点a到结点b之间的关系；用圆括号表示无向关系，如 (a，b) 表示既存在结点a到结点b之间的关系，又存在着结点b到结点a之间的关系。设r是一个D到D的关系，$r \in R$，若d，d′$\in D$，且〈d，d′〉$\in r$，则称d′为d的后继结点，d是d′的前驱结点，这时d和d′是相邻的结点（都是相对r而言的）；如果不存在一个d′，使得〈d，d′〉$\in r$，则称d为r的终端结点；如果不存在一个d′，使得〈d′，d〉$\in r$，则称d为r的开始结点；如果d既不是终端结点也不是开始结点，则称d是内部结点。

例1.4　一个12位的十进制数可以用三个4位的十进制数表示：

6524，3587，1496 —— a_1(6524)，a_2(3587)，a_3(1496)

a_1，a_2，a_3之间存在“次序”关系：〈a_1，a_2〉，〈a_2，a_3〉

很显然，

6524，3587，1496 ≠ 3587，6524，1496，也就是说，

$$a_1\ \ a_2\ \ a_3 \neq a_2\ \ a_1\ \ a_3$$

例1.5　一个2行3列的二维数组{a_1，a_2，a_3，a_4，a_5，a_6}

a_1	a_2	a_3
a_4	a_5	a_6

存在着两种关系，分别是行的次序关系：

$$row = \{\langle a_1, a_2\rangle, \langle a_2, a_3\rangle, \langle a_4, a_5\rangle, \langle a_5, a_6\rangle\}$$

和列的次序关系：

$$col = \{\langle a_1, a_4\rangle, \langle a_2, a_5\rangle, \langle a_3, a_6\rangle\}$$

显然，数组

a_1 a_3 a_5

a_2 a_4 a_6

和

a_1 a_2 a_3

a_4 a_5 a_6

是完全不同的。

该例中同样的数据元素，但是不同的关系构成了不同的（数据）结构。由此可见，数据结构不仅描述了在这个结构中有哪些数据元素，还刻画了这些元素之间的关系。

上述数据结构的定义仅是对操作对象的一种数学描述，换句话说，是从操作对象抽象出来的数学模型。结构定义中的关系是数据元素之间的逻辑关系，因此称之为数据的逻辑结构。讨论数据结构的目的是为了在计算机中实现对它的操作，因此还需研究如何在计算机中表示它。

数据结构在计算机中的表示称为物理结构，又称为存储结构。它是逻辑结构在存储器中的映像，包括数据元素的表示和关系的表示。数据结构的逻辑结构中的数据元素的映像方法是用二进制位（bit）的位串来表示数据元素，如一个十进制数126可以用一个位串1111110来表示，即$(126)_{10}=(176)_8=(1111110)_2$。一个字符“A”可以用其ASCII码01000001这样一个位串来表示，即$A=(65)_{10}=(101)_8=(01000001)_2$。

数据结构的逻辑结构中的关系的映像方法可以有顺序、链式、索引和散列等表示方法。

顺序映像（存储结构）的特点是借助元素在存储器中的相对位置表示数据元素之间的关系。顺序存储结构是一种最基本的存储表示方法，通常借助于程序设计语言中的数组来实现。

例1.6　有一数据结构$G = (D, R)$，其中$D = \{d_1, d_2, d_3, d_4, d_5\}$，$R = \{\langle d_1, d_2\rangle, \langle d_2, d_3\rangle, \langle d_3, d_4\rangle, \langle d_4, d_5\rangle\}$。假定每个结点占一个存储单元，结点$d_1$放在200号单元中，则顺序方式存储的数据结构如图1-4所示。

200	d_1
201	d_2
202	d_3
203	d_4
204	d_5

图1-4　顺序方式存储的数据结构

链式映像（存储结构）是一种非顺序映像方法，借助指示元素存储地址的指针（Pointer）表示数据元素之间的逻辑关系，逻辑上相邻的元素其物理位置不要求相邻。链式存储结构通常借助于程序设计语言中的指针类型来实现。

例1.7　例1.6的数据结构的链式存储表示如图1-5所示。

图中的符号“NIL”表示空指针，该指针不是一个有意义的值，即不表示任何具体结点的单元地址。

从图1-4可以看出，在顺序方式实现的存储中所有的存储空间都被结点数据占用，它是一种紧凑结构。而在链式存储表示中一部分存储空间存放的是表示数据关系的附加信息，即指针，因此是一种非紧凑结构。

存储结构的存储密度定义为结构中数据本身所占的存储量和整个结构所占的存储量之比，即

$$d = \frac{\text{数据本身所占的存储量}}{\text{整个结构所占的存储量}}$$

可见，紧凑结构的存储密度为1，非紧凑结构的存储密度小于1。存储密度越大，则存储空间的利用率越高。但是非紧凑结构中存储的附加信息会给某些运算带来极大的方便，如在进行插入、删除等运算时链式存储结构比顺序存储结构就方便得多。其实非紧凑结构是牺牲了存储空间换取了机器时间。

	info	link
200	d_1	204
201		
202	d_3	203
203	d_4	207
204	d_2	202
205		
206		
207	d_5	NIL

图1-5 链式方式存储的数据结构

在索引存储方式中，线性结构中的数据结点被排成一个序列：d_1，d_2，…，d_n，每个结点d_i在序列里都有对应的位置数i，整个位置数就可以作为结点的索引。索引存储方式就是用结点的顺序号i来确定结点的存储地址。索引存储方式兼有静态和动态特性。

散列存储方式的主要思想是，在记录的存储地址和它的关键字之间建立一个确定的对应关系，使每个关键字和一个唯一的存储位置相对应。方法是根据设定的某个函数 f(key)（散列函数）和处理冲突的方法将一组关键字映像到一个有限的连续地址集（区间）上，并以关键字在地址集中的“映像”作为记录在表中的存储位置，这种表称为散列表或哈希表。

一般数据结构的存储映像都采用这四种基本映像之一，或是它们的组合。同一个逻辑结构可以采用几种不同的映像方法，视问题的不同需求和具体应用而定。

在不同的程序设计环境中，存储结构有不同的描述方法。如果用高级语言进行程序设计，则可用高级程序设计语言提供的数据类型描述存储结构。如例1.4中的例子所示，以三个带有次序关系的整数表示一个长整数时，6524，3587，1496 —— a_1(6524)，a_2(3587)，a_3(1486)，可以用C语言中提供的整数数组类型定义长整数为：

```
typedef int Long_int[3];
```

1.3 抽象数据类型及其表示与实现

算法与数据结构作为一门计算机专业的专业基础课，本身也在不断的发展。一方面，发展各专门领域中特殊的数据结构，如多维图形数据结构；另一方面，从抽象数据类型的观点来讨论数据结构。

首先讨论数据类型（Data Type）的概念。数据类型是对数据的取值范围、数据元素之间的结构以及允许施加操作的一种总体描述。每一种计算机程序设计语言都定义有自己的数据类型。一般有整数、实数（浮点数）、字符、字符串、指针、数组、记录、类和文件等数据类型。例如，整数类型在计算机系统中通常用两个或四个字节表示。若采用两个字节，则整数表示范围在 -2^{15}～$2^{15}-1$，即 -32768～32767之间；若采用四个字节，则整数表示范围在 -2^{31}～$2^{31}-1$，即 -2147483648～2147483647之间。对整数类型的数据允许施加的操作（运算）通常有：单目取正取负运算，双目加、减、乘、除、取模等运算以及双目等于、不等于、大于、大于等于、小于、小于等于等关系（比较）运算以及赋值运算等。字符类型在计算机中通常用一个字节或两个字节表示，无符号表示范围分别在0～255或0～32767之间，能够分别表示至多256或32768种字符的编码。对字符类型的数据允许进行的操作主要为赋值和各种关系运算。字符串类型是

字符顺序排列的线性结构，每一个具体的字符串（其最大长度由具体的语言规定）都是字符串类型中的一个值，对字符串的操作有求串长度、串复制、串连接和串比较等。

按“值”的不同特性，数据类型可以分为简单类型和结构类型两大类。任一种简单类型中的每个数据都是无法再分割的整体，也称为原子类型。如一个整数、实数、字符、指针、枚举值、逻辑值等都是无法再分割的整体。任一种结构类型都是由简单类型数据按照一定的规则构造而成的，并且结构类型仍可以包含结构类型。所以一种结构类型中的数据（即结构数据）可以分解为若干个简单类型数据或结构类型数据，每个结构数据仍可再分。如数组就是一种结构类型，它由若干个分量组成，其中的每个分量可以是整数，也可以是数组等。数组中的每个数据（元素）都可以通过下标运算符直接访问。同样记录也是一种结构类型，它由固定个数的不同（也可以相同）类型的数据按线性结构排列而成，记录中的每个记录值包含有固定个数的不同类型数据，每个数据（域）都可以通过成员运算符直接访问。

无论是简单类型还是结构类型都有“型”和“值”的概念。一种数据类型中的任一数据称为该类型中的一个值（又称为实例），该值（实例）与所属数据类型具有完全相同的结构，数据类型所规定的操作就是在值上进行的。所以在一般的叙述中，并不明确指出是“型”还是“值”，应根据实际情况加以理解，例如，提到记录时，当讨论的是记录结构则认为是记录型，而当讨论的是具体的一条记录时则认为是记录值。

抽象数据类型（Abstract Data Type，ADT）是一个数学模型以及定义在该模型上的一组操作。抽象数据类型包含有一般数据类型的特征，但含义比一般数据类型更广、更抽象。一般数据类型通常由具体语言系统的内部定义，直接提供给用户定义数据并进行相应的运算，因此也称它们为系统预定义数据类型。抽象数据类型通常由用户根据已有的数据类型定义，包括定义其所含数据（数据结构）和在这些数据上所进行的操作。定义抽象数据类型就是定义其数据的逻辑结构和操作说明，而不必考虑数据的存储结构和操作的具体实现（即具体操作代码），从而使得抽象数据类型具有很好的通用性和可移植性，便于用任何一种语言，特别是面向对象的语言实现。

抽象数据类型和上面讨论的数据类型实质上是一个概念。例如，各个计算机系统都拥有的“整数”类型其实也是一个抽象数据类型，因为尽管它们在不同的处理器上实现的方法可能不同，但由于其定义的数学特性相同，所以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。

使用抽象数据类型可以更容易地描述现实世界。例如，用线性表抽象数据类型描述学生成绩表，用树或图抽象数据类型描述遗传关系以及城市道路交通图等。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。也就是说，在进行抽象数据类型设计时，把类型的定义与其实现分离开来。

按抽象数据类型的值的不同特性，抽象数据类型可分解为原子类型和聚合类型两大类。原子类型是其值不可分解的抽象数据类型，如整型数据类型。聚合类型又可分为固定聚合类型和可变聚合类型。其中，固定聚合类型的值由确定数目的成分按某种结构组成，如复数；而可变聚合类型的值的成分数目不确定，例如，可定义一个“有序整数序列”的抽象数据类型，其中序列的长度是可变的。

和数据结构的形式定义相对应，抽象数据类型可用如下三元组表示：

$$(D, R, P)$$

其中D是数据对象，即具有相同特性的数据元素的集合；R是D上的关系集合；P是对D的基本操作集合。

抽象数据类型的定义格式如下：

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

其中的数据对象和数据关系可用伪代码描述，例如，线性表的抽象数据类型可定义如下：

ADT List

{数据对象：$D=\{a_i|a_i\in ElemSet,\ i=1,2,\cdots,n,\ n\geq 0\}$

数据关系：$R=\{\langle a_{i-1},a_i\rangle|a_i,a_{i-1}\in D,\ i=2,\cdots,n\}$

基本操作：

线性表初始化：ListInit (L)；

求线性表的长度：ListLength (L)；

取表元素：ListGet(L，i)；

定位查找：ListLocate (L，x)；

清空线性表：ListClear (L)；

判空线性表：ListEmpty (L)；

求前驱：ListPrior (L，e)；

求后继：ListNext(L，e)；

插入：ListInsert(L，i，e)；

删除：ListDelete(L，i)；

}ADT List

抽象数据类型ADT中的基本操作的定义格式如下：

基本操作名（参数表）

初始条件：〈初始条件描述〉

操作结果：〈操作结果描述〉

“初始条件”描述了操作执行之前数据结构和参数应满足的条件。“操作结果”说明了操作正常完成之后，数据结构的变化状况和应返回的结果。若初始条件为空，则可省略。例如，上述线性表的抽象数据类型中的求线性表的长度操作可定义如下：

ListLength (L)

初始条件：线性表L存在。

操作结果：返回线性表L中所含元素的个数。

抽象数据类型可以通过固有数据类型来表示和实现，即利用处理器中已存在的数据类型来说明新的结构，用已经实现的一些操作组合来实现新的操作。本书采用介于伪代码和C语言之间的类C语言作为描述工具，有时也用伪代码描述一些只含抽象操作的抽象算法。

伪代码是一种用于描述算法的语言。它类似于计算机语言，但不是计算机语言；它是供人们阅读的，不是让计算机执行的。它可以使用源于某种计算机语言的赋值语句、条件语句和循环语句。在算法中我们也可以使用自然语言来描述某一步，只要这步显然能被执行完成。

伪代码语言介于高级程序设计语言和自然语言之间，它忽略高级程序设计语言中一些严格的语法规则与描述细节，因此它比程序设计语言更容易描述和被人理解，而比自然语言更接近程序设计语言。它虽然不能直接执行，但很容易被转换成计算机语言。类C语言是由伪代码和C语言组合而成的一个描述工具，采用了C语言的核心部分，并为描述方便进行了扩充。

1.4　算法和算法分析

1.4.1　算法的定义及特性

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。简单地说算法就是解决特定问题的方法。描述一个算法可以采用文字叙述，也可以采用传统流程图、N-S图或PAD图等，本书采用类C语言描述。

算法与数据结构的关系紧密，在进行算法设计时首先要确定相应的数据结构。如在100个杂乱无章的数中查找一个给定的数，则只能用顺序查找的方法，效率较低。但是如果这100个数已经按照从小到大的顺序排列好的话，则可以采用折半查找的方法，显然这比顺序查找的效率要高得多。

一个算法具有下列重要特性。

1) 有穷性：算法只执行有限步，并且每步应该在有限的时间内完成。这里“有限”的概念不是纯数学的，而是在实际上是合理的和可接受的。如一个简单的算法程序不应该在一个月还没有完成。

下面的算法不符合有穷性。

```
loopforever
   { while(1)
     printf("do nothing");
    }
```

2) 确定性：算法中的每一条指令必须有确切的含义，无二义性。在任何条件下，算法只有唯一的一条执行路径，即对于相同的输入只能得出相同的输出。

3) 可行性：算法中描述的操作都必须足够基本，即都是可以通过已经实现的基本运算执行有限次来实现的。

例如，“把两个变量交换”和“将变量a的值增1”的算法，就是足够基本的，是可行的。而“把两个变量a和b的最大公因子s送给变量c”的算法就不是足够基本的，是不可行的。

4) 输入：算法具有零个或多个输入，也就是说，算法必须有加工的对象。输入取自特定的数据对象的集合。输入的形式可以是显式的，也可以是隐式的，有时候输入可能被嵌入在算法中。

5) 输出：算法具有一个或多个输出。这些输出与输入之间有某种确定的关系。这种确定的关系就是算法的功能。例如：

```
int getsum(int num)
     {int  sum=0;
      for (i=1;i<=num; i++)
        sum+=i;
      return sum;
     }
```

无输出的算法没有任何意义。

算法的含义和程序十分类似，但是也有区别。一般来说，一个程序并不需要满足上述的第一个条件（有穷性）。例如操作系统程序，只要整个系统不遭破坏，操作系统程序就永不结束。另外，程序是用机器可执行的语言书写的，而算法通常并没有这种限制。

1.4.2　算法的设计要求

要设计一个好的算法通常要考虑达到以下要求。

1) 正确性（Correctness）：算法的执行结果应当满足预先规定的功能和性能要求。正确性是设计和评价一个算法的首要条件，如果一个算法不正确，即不能完成所要求的任务，其他方面也就无从谈起。一个正确的算法是指在合理的数据输入下，能够在有限的运行时间内得出正确的结果。通过采用各种典型的输入数据上机反复调试算法，使得算法中的每段代码都被测试过，若发现错误及时纠正，最终可以验证出算法的正确性。当然，要从理论上验证一个算法的正确性，并不是一件容易的事，也不属于本课程所研究的范围，故不进行讨论。

2) 可读性（Readability）：是指算法的可读性程度。算法主要是为了人的阅读与交流，其次才是为了让计算机执行。因此算法应该思路清晰、层次分明、简洁明了、易读易懂，必要的地方加以注释。此外，晦涩难懂的程序易于隐藏较多的错误而难以调试和修改。可读性还要求对算法中出现的各种自定义变量和类型做到“见名知义”，即读者一看到变量（或类型名）就知道其功用。总之，算法的可读性不仅能让读者理解算法的设计思想，同时也可以方便算法的维护。

3) 健壮性（Robustness）：是指一个算法对不合理（又称不正确、非法、错误等）数据输入的反应和处理能力。一个好的算法应该能够识别错误数据并进行相应的处理。对错误数据的处理一般包括打印错误信息、调用错误处理程序、返回标识错误的特定信息、终止程序运行等。经过对错误输入数据的适当处理，可以避免引起严重后果。例如，数组都是有上下界的，当访问的下标越过该界限时，系统应该出现保护性错误，如直接返回一个错误指示，以避免“数组越界的错误”。

4) 高效性（High Efficiency)：算法应有效使用存储空间和有较高的时间效率。两者都与问题的规模有关。

1.4.3 算法效率的衡量方法及其准则

一个问题可能存在着多种解法，而算法设计者需要在所花费的时间和所使用的空间资源两者之间采取折衷，通过算法分析，可以判断所提出的算法是否现实。进行算法性能分析的目的是寻找高效的算法来解决问题，提高工作效率。

衡量算法效率的方法主要有两大类：算法的事后统计方法（后期测试）和算法的事前分析估算方法。

1) 事后统计方法：利用计算机的时钟进行算法执行时间的统计。在算法中的某些部位插装时间函数time()测定算法完成某一功能所花费的时间。这种方法有非常明显的缺陷，即首先必须把算法转变成为程序执行，其次进行时间统计时依赖于硬件和软件环境，这容易掩盖算法本身的优劣。

2) 事前分析估算方法：用高级语言编写的程序运行的时间主要取决于下列因素。

- 算法选用的策略。
- 问题的规模。随着处理问题的数据增大，处理会越来越困难复杂，把描述数据增大程度的量叫做问题规模。规模越大，消耗时间越多。例如，求100以内的素数和求10000以内的素数的执行时间显然是不同的。
- 编写程序的语言。对于同一个算法，实现语言的级别越高，其执行效率就越低。
- 编译程序所产生的目标代码的质量。对于代码优化较好的编译程序，其所生成的程序质量较高。
- 机器执行指令的速度。

显然，上述后面三条与算法设计是无关的。也就是说，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，效率均不相同。这表明使

用绝对的时间单位衡量算法的效率是不合适的。除去这些与计算机硬件、软件有关的因素，可以认为一个特定算法的“运行工作量”的大小只依赖于问题的规模（通常用整数量 n 来表示），或者说，它是问题规模的函数。

一个算法的时间复杂度（Time Complexity）$T(n)$是该算法的时间耗费，是该算法所求解问题规模n的函数。当问题规模n趋向无穷大时，时间复杂度$T(n)$的数量级（阶）称为算法的渐近时间复杂度（Asymptotic Time Complexity），可记作

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

它表示随问题规模n的增大，算法执行时间的增长率和函数$f(n)$的增长率是相同的。

例1.8　假定有两个算法A和B求解同一个问题，它们的算法时间复杂度分别是$T_A(n) = 100n^2$，$T_B(n) = 5n^3$。因此有：

1) 当输入量较小（如$n < 20$）时，有$T_A(n) > T_B(n)$，后者花费的时间较少。

2) 随着问题规模n的增大，两个算法的时间消耗之比$5n^3/(100n^2) = n/20$也随着增大。也就是说，当问题规模较大时，算法A比算法B要有效得多。

上述算法A和B的渐进时间复杂度分别为$O(n^2)$和$O(n^3)$，它们从宏观上评价了这两个算法在时间方面的质量。在具体进行算法分析时，往往对算法的时间复杂度和渐进时间复杂度不予区分，而经常将渐进时间复杂度$T(n) = O(f(n))$简称为时间复杂度。

算法主要由程序的控制结构（顺序、分支、循环）和原操作（固有数据类型的操作，是必需的操作）构成，算法的时间主要取决于两者。具体而言，从算法中选取一种对于所研究的问题来说是基本操作的原操作，以该基本操作在算法中重复执行的次数作为算法运行时间的衡量准则。多数情况下基本操作就是最深层循环内的语句中的原操作，它的执行次数和包含它的语句的频度相同。语句的频度（Frequency Count）指的是该语句重复执行的次数。

例1.9　赋值语句。

```
{ ++x; s=0; }
```

该算法含基本操作“x增1”的语句的频度为1，则时间复杂度为$O(1)$，为常量阶。

例1.10　简单的循环。

```
for(i=1;i<=n;++i)
   {++x;s+=x; }
```

该算法含基本操作“x增1”的语句的频度为n，则时间复杂度为$O(n)$，为线性阶。

例1.11　双重循环。

```
for(j=1;j<=n;++j)
    for(k=1;k<=n;++k)
        {++x;s+=x;}
```

该算法含基本操作“x增1”的语句的频度为$n \times n$，则时间复杂度为$O(n^2)$，为平方阶。

但是，并非所有双重循环的时间复杂度都是$O(n^2)$。

```
for(j=1;j<=n;j*=2)
    for(k=1;k<=n;++k)
        { ++x;s+=x; }
```

外层循环每循环一次j就乘以2，直至$j > n$，所以共执行$\lceil \log_2 n \rceil$次。而内层循环执行次数总是为n，总的时间代价为

$$\sum_{j=1}^{\lceil \log_2 n \rceil} \sum_{k=1}^{n} = \sum_{j=1}^{\lceil \log_2 n \rceil} n = n \log_2 n$$

所以时间复杂度为$O(n\log_2 n)$，为线性对数阶。

例1.12 矩阵相乘。

```
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] = c[i][j]+a[i][k]* b[k][j];
     }
```

该算法中，第3层循环中的循环体执行次数最多为n^3，所以该算法的时间复杂度是$O(n^3)$，为立方阶。

例1.13 直接选择排序。

```
void   SelectSort(RecType R[],int n)
{ //对记录序列R[1..n]作直接选择排序
  for(i=1; i<n; i++)
    { //选择第i小的记录,并交换到排序后的正确位置
      k=i;//假定第i个元素的关键字最小
      for(j=i+1;j<=n;j++)              //找最小元素的下标
        if(R[j].key<R[k].key)  k=j;
      if(i!=k)  R[i]←→R[k];            //与第i个记录交换
     }//for
}//SelectSort
```

算法中的“比较”操作为基本操作。无论记录的初始排列如何，所需进行的关键字的比较次数都相同，均为：

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

在最好情况下，即待排序记录初始为正序时，交换次数为0次；在最差情况下，即待排序记录初始为反序时，发生了$(n-1)$次交换（$3(n-1)$次赋值）。所以直接选择排序的时间复杂度为$O(n^2)$。

最坏情况下的时间复杂度称为最坏时间复杂度，在不作特别说明时，我们讨论的时间复杂度是最坏情况下的时间复杂度。

例1.14 起泡排序。

```
void   BubbleSort(RecType R[],int n)
{ //起泡排序
  i = n; //i 指示无序序列中最后一个记录的位置
  while(i>1)
    { lastExchange=1;                                  //记录最后一次交换发生的位置
      for(j=1;j<i;j++)
        if(R[j].key>R[j+1].key)
          { temp=R[j];R[j]=R[j+1];R[j+1]=temp;     //逆序时交换
            lastExchange=j;
           }//if
      i=lastExchange;
   }//while
}//BubbleSort
```

待排序序列正序时为最好的情况，只需要进行一趟起泡，比较次数为$n-1$次，交换次数为0次。待排序序列为逆序时为最坏的情况，需要进行$n-1$趟起泡，比较次数为

$$\sum_{i=n}^{2}(i-1)=n(n-1)/2$$

交换次数为 $\frac{1}{2}n(n-1)$，即 $\frac{3}{2}n(n-1)$ 次赋值。

因此，起泡排序算法的时间复杂度为$O(n^2)$。

例1.15　用百元买百笔。已知钢笔3元一支，圆珠笔2元一支，铅笔5角一支。给出解决方案。

可以用穷举法求出：

```
for( i = 1; i < =100; i++)
   for( j = 1; j < =100; j++)
       for( k= 1; k< =100; j++)
          if(i+j+k==100 && 3*i+2*j+0.5*k==100)
               printf("i=%d,j=%d,k=%d",i,j,k)
```

但是事实上为了要买到总共100支笔，最多只可以买20支钢笔、34支圆珠笔，100减去钢笔和圆珠笔的总数就是可以买的铅笔的总数，因此可以写出如下算法：

```
for( i = 1; i < =20; i++)
   for( j = 1; j < =34-i; j++)
     if(3*i+2*j+(100-i-j) *0.5==100)
          printf("i=%d,j=%d,k=%d",i,j,100-i-j);
```

对上述两个算法进行测试，第一个算法内层循环超过100万次，在某机器上运行了50min；第二个算法的if语句执行了525次，在相同的机器上仅运行了2s，两个算法的运行时间竟相差了1500倍。

由该例可以看到，对于同样的问题，算法的设计是何等重要。

常见的时间复杂度及其关系如下：

$$O(1)<O(\log_2 n)<O(n)<O(n\log_2 n)<O(n^2)<O(n^3)<\cdots<O(n^k)<O(2^n)<O(3^n)<O(n!)\cdots$$

1.4.4　算法的存储空间需求

空间复杂度（Space Complexity）或称为空间复杂性是指解决问题的算法在执行时所占用的存储空间。它也是衡量算法有效性的一个指标，记作

$$S(n)=O(g(n))$$

其中n为问题的规模（或大小）。表示随着问题规模n的增大，算法运行所需存储量的增长率与函数$g(n)$的增长率相同。

算法的存储量包括三个部分：程序本身所占的存储空间、输入数据所占的空间以及辅助变量所占的空间。再具体一些，也就是进行程序设计时，程序的存储空间、变量占用空间、系统堆栈的使用空间等。也正由此，空间复杂度的度量分为两个部分：固定部分和可变部分。存储空间的固定部分包括程序指令代码的空间，常数、简单变量、定长成分（如数组元素、结构成分、对象的数据成员等）变量所占空间等。可变部分包括与实例特性有关的成分变量所占空间、引用变量所占空间、递归栈所占空间以及通过malloc等命令动态使用的空间等。

如果输入数据所占空间只取决于问题本身，和算法无关，则在讨论算法的空间复杂度时，只需分析除输入和程序之外的辅助变量所占的额外空间即可。如果所需额外空间相对于输入数据量来说只是一个常数，则称此算法为“原地工作”，此时的空间复杂度为$O(1)$；如果算法

所需的存储量与特定的输入有关，那么同时间复杂度一样，也是按照最坏的情况进行考虑。

对于一个算法，其时间复杂度和空间复杂度往往是相互影响的，当追求一个较好的时间复杂度时，可能会使空间性能变差，即可能导致较多的存储空间。反之，当追求一个较好的空间复杂度时，可能会使时间性能较差，即可能导致占用较长的运行时间。另外，算法的所有性能之间都存在着或多或少的相互影响。因此，当设计一个算法（特别是大型算法）时，需要综合考虑算法的各项性能、算法的使用频率、算法处理的数据量的大小、算法描述语言的特性、算法运行的机器系统环境等诸多因素，通过权衡利弊才能够设计出理想的算法。

1.5 类C语言描述

本书采用类C语言描述抽象数据类型和算法。类C语言是由伪代码和C语言组合而成的一个描述工具，采用了C语言的核心部分，并为描述方便进行了扩充。下面是类C语言的主要说明：

1. 数据结构的存储结构

数据结构的存储结构用类型定义（typedef）描述。数据元素类型的定义为ElemType，由用户在使用该数据类型时自行定义。例如：

```
typedef int ElemType ;
```

说明用ElemType 表示**int**类型。

利用define 定义布尔常量：

```
#define TRUE 1
#define FALSE 0
```

2. 基本操作

算法中的基本操作用以下形式的函数描述：

```
函数类型 函数名（函数参数表）
{//算法说明
  语句序列
}//函数名
```

函数的参数需要说明类型，算法中使用的辅助变量可以不声明变量类型。

3. 赋值语句

1) 简单赋值：变量名=表达式；

2) 串联赋值：变量名1=变量名2= … =变量名*k*=表达式；

3) 成组赋值：（变量名1，…，变量名*k*）=（表达式1，…，表达式*k*)；

结构名=结构名；

结构名=(值1，…，值*k*)；

变量名[ ]=表达式；

变量名[起始下标..终止下标]=变量名[起始下标..终止下标]；

4) 交换赋值：变量名↔变量名；

5) 条件赋值：变量名=条件表达式？表达式T：表达式F；

4. 选择语句

1) **if**（表达式）语句；

2) **if**（表达式）语句；

else 语句；

3) **switch**(表达式)

{ **case** 值1：语句序列1；**break**；

……

case 值*n*：语句序列*n*；**break**；

default：语句序列*n*+1；

}

4) **switch**

{ **case** 条件1：语句序列1；**break**；

……

case 条件*n*：语句序列*n*；**break**；

default：语句序列*n*+1；

}

5. 循环语句

1) **for**（赋初值表达式；条件；修改表达式序列） 语句；

2) **while**（条件）语句；

3) **do**{ 语句序列}**while**（条件）；

6. 结束语句

1) 函数结束语句：**return**（表达式）；

return;

2) **case**结束语句：**break**;

3) 异常结束语句：**exit**（异常代码）；

7. 输入和输出语句

1) 输入语句：scanf（变量1，变量2，…，变量*n*）；

2) 输出语句：printf（变量1，变量2，…，变量*n*）；

8. 注释

1) 多行注释：/* 注释内容*/

2) 单行注释：//文字序列

9. 基本函数

max（表达式1，…，表达式*n*），min，abs，floor，ceil，eof，eoln

10. 逻辑运算约定

1) &&：与运算符

2) ||：或运算符

3) !：非运算符

习题

一、基础知识题

1.1　简述下列概念：

数据，数据元素，数据类型，数据结构，逻辑结构，存储结构，算法。

1.2　数据的逻辑结构分哪几种，为什么说逻辑结构是数据结构的主要方面？

1.3　试举一个数据结构的例子，叙述其逻辑结构、存储结构、运算三方面的内容。

1.4　简述算法的五个特性以及对算法设计的要求。

1.5　设*n*是正整数，求下列程序段中带@记号的语句的执行次数。

(1)
```
i=1;k=0;
  while(i<n)
    {k=k+50*i; i++;  @
    }
```

(2)
```
i=1;j=0;
  while(i+j<=n)
    {if(i>j)j++;       @
          else i++;  } @
```

(3)
```
x=y=0;
  for(i=0;i<n;i++)    @
  for(j=0;j<n;i++)    @
    {x++;             @
     for(k=0;i<n;i++) @
       y++;           @
     }
```

(4)
```
x=91;y=100;
  while(y>0)
     if(x>100)
     {x=x-10; y--;     @
     }
  else x++;            @
```

1.6 有实现同一功能的两个算法A_1和A_2，其中A_1的时间复杂度为$T_1 = O(2^n)$，A_2的时间复杂度为$T_2 = O(n^2)$，仅就时间复杂度而言，请具体分析这两个算法哪一个好。

1.7 选择题：算法分析的目的是(　　)

A. 找出数据结构的合理性　　B. 研究算法中输入和输出的关系

C. 分析算法的效率以求改进　　D. 分析算法的易懂性和文档特点

二、算法设计题

1.8 已知输入x，y，z三个不相等的整数，设计一个“高效”算法，使得这三个数按从小到大输出。“高效”的含义是用最少的元素比较次数、元素移动次数和输出次数。

1.9 在数组$A[n]$中查找值为k的元素，若找到则输出其位置i（$1 \le i \le n$），否则输出0作为标志。设计算法求解此问题，并分析在最坏情况下的时间复杂度。