# 第 2 章　数据结构与算法概述

著名计算机科学家，Pascal 语言发明者 N·沃思（Niklaus Wirth)教授在 1976 年提出了著名公式：算法+数据结构 = 程序。因此要学好程序设计，对数据结构和算法有一定了解是必需的。程序就是在数据的某些特定的结构和表示的基础上对于算法的描述，算法与数据结构都是计算机程序的两个最基本的概念，是程序设计中相辅相成、不可分割的两个方面。数据结构是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的集合；算法是对特定问题求解步骤的一种描述，是由指令的有序序列组成。

本章对数据结构和算法的基本概念进行简要的介绍。

## 2.1　引　言

N. 沃思教授提出的著名公式（算法 + 数据结构 = 程序）。说明一个程序应该包括两个方面的内容：一是对数据的描述，在程序中要指定数据的类型和数据的组织形式，即数据结构。二是对操作的描述，即操作步骤，也就是算法。

数据是操作的对象，操作的目的是对数据进行加工处理，以得到期望的结果。

准确地说，一个程序规定了某个数据结构上的一个算法。算法是程序的核心，它在程序设计乃至在整个计算机科学中都占据重要地位。

比如说程序设计就像盖房子，数据结构就像砖、瓦，而算法就是设计图纸。你若想盖房子首先必须有原料（数据结构），但是这些原料不能自动地盖起了你想要的房子，你必须按照设计图纸（算法）上的说明一砖一瓦地去砌。这样你才能拥有你想要的房子。

程序设计也一样，你使用的编译工具（如 Jave /C/Basic/Pascal 等）中有各种功能语句或基本结构（如 Read/Write/Real/Boolean 等），它们不会自动排列成你想要的程序代码。你得按照程序规定的功能去编写，而程序功能的实现就是算法的具体体现。所以通俗地说，你必须按照特定的规则，把特定的功能语句和基本结构按照特定的顺序排列起来，形成一个有特定功能的程序。

数据结构是程序设计这座大厦的基础，没有基础，无论设计有多么高明，这座大厦不可能建造起来的。算法则是程序设计的思想，是它的灵魂，没有灵魂的程序不能叫程序，只是一堆杂乱无章的符号而已。在程序设计中，数据结构就像物质，算法就是意识。这就像哲学上说的：意识是依赖于物质而存在的，物质是由于意识而发展。双方是相互依存、缺一不可的。

数据结构内容不多，仅仅由几个系统自带的基本结构（像顺序结构、分支结构、循环结构、函数过程），基本数据类型（整型、实型、布尔型、字符型等）和用户定义的高级点的数据结构[数组、集合、文件、指针结构（队列、栈、树、图等）……]组成，就这么多。可是算法却不同了，它是多种多样的，它可以使数据以你想要的方式排列（当然要符合语法和功能要求）。再打个比方，数据结构是人体的各种组织、器官，算法则是人的思想。你可以用你的思想去支配你身体的各个可以运动的器官随意运动。例如，你想去取一个苹果，你可以走过去，也可以跑过去，甚至可以爬过去。但是无论如何，你的器官还是你的器官（没有变），目的还是同一个目的（取苹果），而方式却是随心所欲。这就是算法的灵活性，不固定性。因此可以这样说：数据结构是“死”的，而算法是“活”的。

例如，我们编程求 1+2+3+…+100，可以先进行 1 加 2，再加 3，再加 4，一直加到 100，而有的人采取这样的方法：

$$\sum_{n=1}^{100} n = 100 + (1+99) + (2+98) + \cdots + (49+51) = 100 + 49 \times 100 + 50 = 5050$$

还可以有其他的方法，比如直接用数列公式等。

当然，方法有优劣之分。有的方法只需进行很少的步骤，而有些方法则需要较多的步骤。一般说，希望采用方法简单、运算步骤少的方法。因此，为了有效地进行解题，不仅需要保证算法正确，还要考虑算法的质量，选择合适的算法。

## 2.2 数据结构概述

计算机科学技术是一门研究信息表示和处理的科学，它不仅涉及算法和程序的结构，同时也涉及程序的加工对象——数据的结构。在计算机科学中，数据结构是计算机中存储、组织数据的方式。通常情况下，精心选择的数据结构可以带来最优效率的算法。

随着计算机的发展和应用，计算机的应用已不再局限于科学计算，而更多地用于控制、管理及数据处理等非数值计算的处理工作。因此这里所说的“数据”是对客观事物的符号表示，是指所有能输入到计算机中并被计算机程序处理的符号的集合。对计算机科学而言，数据的含义极为广泛，如图像、声音等信息都可以通过编码而归之于数据的范畴。数据集合中的每一个个体称为数据元素，是数据的基本单位。

那么，什么是数据结构呢？简单地说，数据结构是一门研究非数值计算的程序设计中计算机的操作对象以及它们之间的关系和操作的学科。它研究的内容包括以下三个方面：

### 1. 逻辑结构

数据的逻辑结构是指数据元素之间的逻辑关系，它只抽象地反映数据元素间的相互关系，而不考虑数据在计算机中的具体存储方式，是独立于计算机的。就数据的逻辑结构而言，根

据元素之间关系的不同特性，通常有 4 种基本结构：

集合：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。

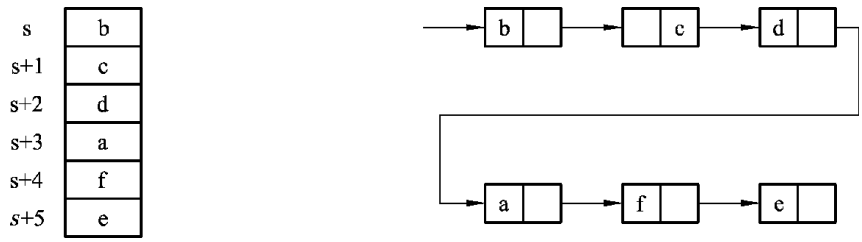
线性结构：结构中的数据元素间存在一对一的关系。

树形结构：结构中的数据元素间存在一对多的关系。

图状结构（或称网状结构）：结构中的数据元素间存在多对多的关系。

### 2. 存储结构

数据的存储结构是指数据在存储器中的存储方式。数据的存储结构也可以说是逻辑结构在计算机存储设备上的物理实现，有时也被称为数据的物理结构。数据存储结构的基本组织方式有顺序存储结构和链式存储结构两种。顺序存储结构是使用一组地址连续的存储单元来存放数据元素，借助元素在存储器中的相对位置来表示数据元素的逻辑关系，如图 2-1（a）所示；而链式存储结构是使用不一定连续的存储单元来存放数据元素，借助指示元素存储地址的指针来表示数据元素之间的逻辑关系，如图 2-1（b）所示。




（a）顺序存储结构　　（b）链式存储结构

图 2-1　数据存储结构

### 3. 基本操作

用户创建一种数据结构，需要定义能在其上进行的操作。可建立于数据结构之上的操作种类很多，常用的有以下几种：建立、销毁、插入元素、删除、更新、查找、遍历等操作。

## 2.3　几种常见的数据结构

### 2.3.1　线性表

#### 1. 线性表的定义

线性表是最简单、最常用的一种数据结构。一个线性表是 $n$ 个数据元素的有限序列。至于每个数据元素的具体含义，在不同的情况下各不相同，它可以是一个数、一个符号，也可以是一条记录信息。

例如，26 个英文字母的字母表（A，B，C，D，…，Z）是一个线性表，表中的数据元素是单个字符。同一线性表中的元素必须具有相同的特性。

又如，某校从 1978 年到 1983 年各种型号的计算机拥有量的变化情况，可以用线性表的形式给出：

(6，17，28，50，92，188)

表中的数据元素是整数。

在稍复杂的线性表中，一个数据元素可以由若干个数据项（item）组成。在这种情况下，常把数据元素称为记录（record），含有大量记录的线性表又称文件（file）。

例如，一个学校的学生健康情况如表 2-1 所示，表中每个学生的情况为一个记录，它由姓名、学号、性别、年龄、班级和健康状况等六个数据项组成。

表 2-1　学生健康情况登记表

| 姓名 | 学号 | 性别 | 年龄 | 班级 | 健康状况 |
|---|---|---|---|---|---|
| 王明 | 10001 | 男 | 18 | 信 1 | 良好 |
| 李四 | 10002 | 女 | 20 | 信 1 | 良好 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

线性表中元素的个数 $n$（$n\geq 0$）定义为线性表的长度，$n=0$ 时称为空表。在非空表中的每个数据元素都有一个确定的位置，如 $a_1$ 是第一个数据元素，$a_n$ 是最后一个数据元素，$a_i$ 是第 $i$ 个数据元素，称 $i$ 为数据元素 $a_i$ 在线性表中的位序。

线性表是一个相当灵活的数据结构，它的长度可根据需要增长或缩短，即对线性表的数据元素不仅可以进行访问，还可进行插入和删除等。

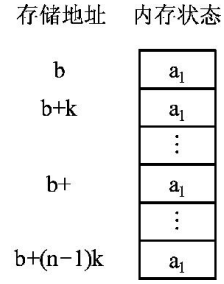



图 2-2　线性表的顺序存储

## 2. 线性表存储结构

线性表存储结构既可以顺序存储也可以链式存储。假设线性表的每个元素 $a_i$ 需占用 $k$ 个存储单元，并以所占的第一个单元的存储地址作为数据元素的存储位置。则顺序存储示意如图 2-2 所示。

链式存储则用一组任意的存储单元存储线性表的数据元素，如图 2-3 所示。

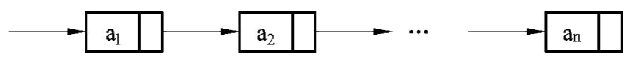


图 2-3　线性表的链式存储

## 2.3.2　栈和队列

### 1. 栈

栈又称为堆栈，是一种操作受限的线性表，是限制在表的一端进行插入和删除的线性表。插入、删除的一端称为栈顶，另一端称为栈底，不含数据元素的栈称为空栈。它的工作方式是“先进后出”，如图 2-4 所示。

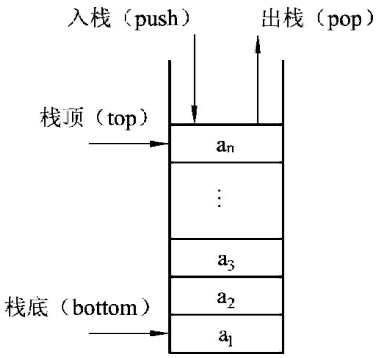


图 2-4　栈的示意图

例如，我们冬天取暖的煤炉，只能从煤炉的顶部依次放入蜂窝煤，最先放入的蜂窝煤在最底部，其他蜂窝煤依次堆放。当蜂窝煤燃烧完后，我们也只能首先把最上面的蜂窝煤开始取出，然后依次取出直到最先放入的蜂窝煤最后取出，这就遵循了栈的后进先出的特点。

又如括号匹配问题：假设表达式中允许包含两种括号：圆括号和方括号，其嵌套的顺序随意，即“( [ ]( ) )”或“[( [ ] [ ] ) ]”等为正确的格式，“[( ] )”或“( [( ) )”或“( ( ) ])”均为不正确的格式。检验括号是否匹配的方法可用“期待的急迫程度”这个概念来描述。

例如考虑下列括号序列：[　(　[　]　[　]　)　]

对应　1　2　3　4　5　6　7　8

当计算机接受了第 1 个括号后，它期待着与其匹配的第 8 个括号的出现，然而等来的却是第 2 个括号，此时第一个括号“[”只能暂时靠边，而迫切等待与第 2 个括号相匹配的、第 7 个括号“ )”的出现，类似地，因等来的是第 3 个括号“[”，其期待匹配的程度较第 2 个括号更急迫，则第 2 个括号也只能靠边，让位于第 3 个括号，显然第 2 个括号的期待急迫性高于第 1 个括号；在接受了第 4 个括号之后，第 3 个括号的期待得到满足，消解之后，第 2 个括号的期待匹配就成为当前最急迫的任务了，……，依次类推。可见，这个处理过程恰与栈的特点相吻合。由此，在算法中设置一个栈，每读入一个括号，若是右括号，或者使置于栈顶的最急迫的期待得以消解，或者是不合法的情况；若是左括号，则作为一个新的更急迫的期待压入栈中，自然使原有的在栈中的所有未消解的期待的急迫性都降了一级。另外，在算法的开始和结束时，栈都应该是空的。

### 2. 队　列

队列是一种先进先出(FIFO)的线性表，它允许在表的一端进行插入，而在另一端进行删除操作，其特点表现为“先进先出”。允许插入的一端称为队尾，允许删除的一端称为队首，如图 2-5 所示。

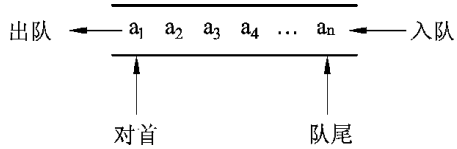


图 2-5　队列示意图

例如，我们去银行办业务，都需要领号排队，这

就遵循了先来先到的顺序，你先来了先办业务，后来后办业务。

又如操作系统中的作业排队。在允许多道程序运行的计算机系统中，同时有几个作业运行。如果运行的结果都需要通过通道输出，那就要按请求输出的先后次序排队。每当通道传输完毕可以接受新的输出任务时，队头的作业先从队列中退出做输出操作。凡是申请输出的作业都从队尾进入队列。

### 2.3.3 树

树型结构是一类重要的非线性数据结构，其中以树和二叉树最为常用，直观地看来，树是以分支关系定义的层次结构。树型结构在客观世界中广泛存在，如人类社会的族谱和各种社会组织都可以用树来形象表示，都可以形象的看做一棵树倒置的形状。树根作为根结点，树的各个分支看做结点，树的叶子看做叶结点，这样就便于理解。

#### 1. 树的定义

树是 $n$（$n \geq 0$）个结点的有限集 $T$，在任意一棵非空树中：

（1）有且仅有一个特定的称为根的结点。

（2）当 $n>1$ 时，其余的结点可分为 $m$（$m \geq 0$）个互不相交的有限集合 $T_1$，$T_2$… $T_m$，其中每个子集都是一棵树，并称其为子树，如图 2-6 所示的一棵树。

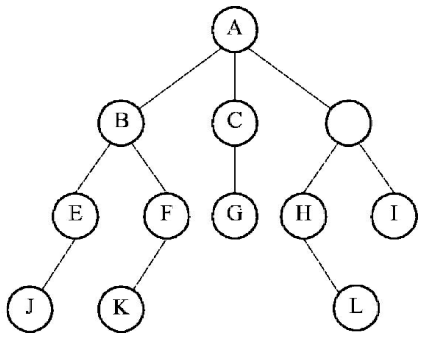



图 2-6 树

#### 2. 基本概念

（1）根结点：无前驱，例如 A 是根结点。

（2）度：在树中，结点拥有的子树的数目称为该结点的度，例如 A 的度为 3。

（3）叶子：度为 0 的结点称为叶子，树中 J、K、G、L、I 都是叶子。

（4）孩子：结点的子树的根称为该结点的孩子。相应地，该结点称为孩子的双亲。B、C、D 是 A 的孩子，A 是 B、C、D 的双亲。同层结点称为兄弟，B、C、D 是兄弟。

（5）祖先：结点的祖先是从根到该结点所经分支上的所有结点。

（6）子孙：以某结点为根的子树中的任一结点都称为该结点的子孙。

（7）结点的层：树根为第一层，根的直接后继结点为第二层，依次类推，这是结点的层次概念。

（8）树的高度：树中结点的最大层次称为树的高度（或者深度）。

如果一棵树的每个结点至多只有两棵子树，且有左右之分，称该树为二叉树。如图 2-7 所示为二叉树。除最后一层无任何子节点外，每一层上的所有结点都有两个子结点（最后一层上的无子结点的结点为叶子结点）。也可以这样理解，除叶子结点外的所有结点均有两个子结点。如果所有叶子结点必须在同一层上，则称该树为满二叉树。

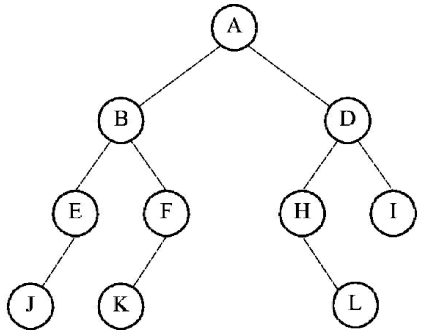
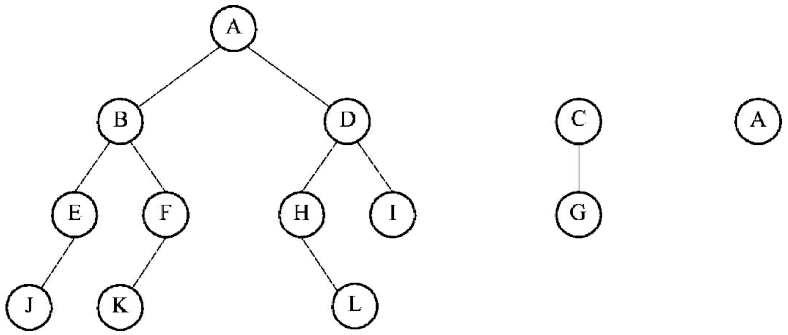



图 2-7　二叉树

多棵树的集合称为森林，如图 2-8 所示。树、森林、二叉树之间可以互相转换。



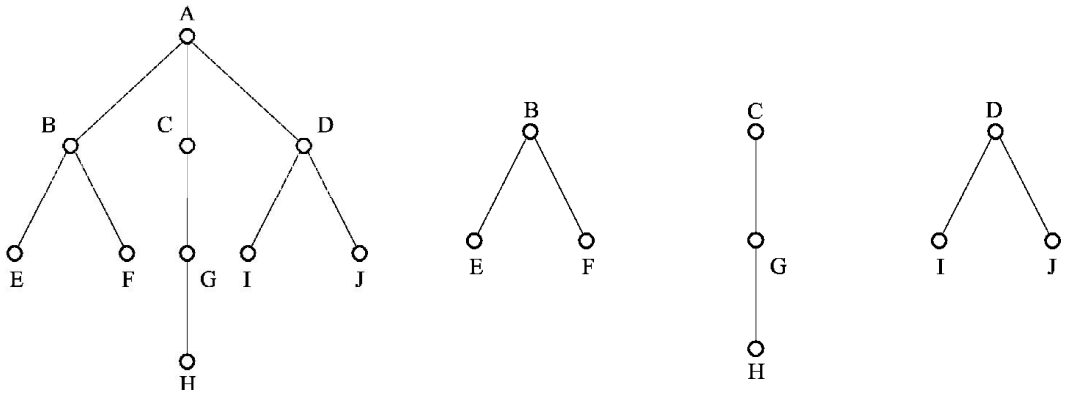

（a）　　（b）　　（c）

图 2-8　森 林

【例 2-1】　一棵树的逻辑结构为 $T=(K,R)$，其中 $K=\{A,B,C,D,E,F,G,H,I,J\}$，$R=\{r\}$，$r=\{<A,B>,<A,C>,<A,D>,<B,E>,<B,F>,<C,G>,<D,I>,<D,J>,<G,H>\}$。请用树形表示法画出此树，并按根将树划分为子树，指出哪个结点是根，哪些结点是树叶，确定每个结点的层数和度数。最后指出树的高度。

解：（1）树形图如图 2-9 所示。

（2）以 A 为根的子树如图 2-10 所示。



（a）　　（b）　　（c）

图 2-9　树形图　　图 2-10　子树集

（3）此树的树根是 A；树叶是 E，F，H，I，J。

（4）此树各结点的层数为：

| | |
|---|---|
| A | 0 层； |
| B，C，D | 1 层； |
| E，F，G，I，J | 2 层； |
| H | 3 层。 |

此树各结点的度数为：

| | |
|---|---|
| A | 3 度； |
| B，D | 2 度； |
| C，G | 1 度； |
| E，F，H，I，J | 0 度。 |

（5）树的高度为 3。

### 2.3.4　图

图是一种较线性表和树更为复杂的数据结构，可以把树看成是简单的图。图的应用极为广泛，在语言学、逻辑学、人工智能、数学、物理、化学、计算机领域以及各种工程学科中有着广泛的应用。

图是由顶点集 $V$ 和边集 $E$ 构成，记作：$G=(V,E)$。

其中，$V$ 是图中顶点的非空有穷集合，$E$ 是两个顶点之间关系的集合，它是顶点的有序或无序对，记作$<v_i, v_j>$或 $(v_i, v_j)$。如图 2-11，图 2-12 所示，它是由顶点和边构成的，顶点集 $V=\{A,B,C,D,E,F\}$，边的集合 $E=\{(A,B),(A,C),(A,D),(B,E),(C,E),(C,F),(D,F),(E,F)\}$。若图中每条边都是有向边，则称 G 为有向图，如图 2-11 所示。否则称为无向图，如图 2-12 所示。

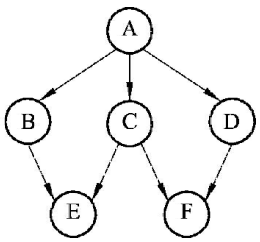



图 2-11　有向图

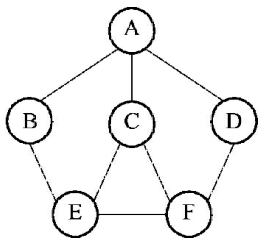



图 2-12　无向图

【例 2-2】　江西省计划在南昌、九江、上饶、吉安之间修建新的高速公路，主要包括九江到南昌，上饶到九江，吉安到上饶，吉安到九江，南昌到吉安。请把城市作为顶点，高速公路作为边用无向图描述。

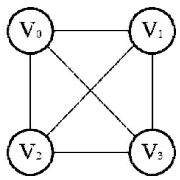


图 2-13

解：用符号表示城市南昌：$v_0$，九江：$v_1$，上饶：$v_2$，吉安：$v_3$。可以描述为有一无向图为 $G=(V,E)$。其中 V 是顶点的集合，E 是边的集合。如图 2-13 所示：

其中：$V=\{v_0,v_1,v_2,v_3\}$，$E=\{<v_1,v_0>,<v_2,v_1>,<v_3,v_2>,<v_3,v_1>,<v_0,v_3>\}$

## 2.4　算法概述

### 2.4.1　什么是算法

计算机解题一般可分解成若干操作步骤，通常把完成某一任务的操作步骤称为求解该问题的算法。程序就是用计算机语言描述的算法。

算法是指完成一个任务所需要的具体步骤和方法。也就是说给定初始状态或输入数据，能够得出所要求或期望的终止状态或输出数据。

既然算法是解决给定问题的方法，算法所处理的对象就是该问题所涉及的数据。程序的目的就是加工数据，而如何加工数据就是算法的目的。

例如给定两个正整数 $m$ 和 $n$，求它们的最大公约数。在学数学的时候，我们都知道最大公约数即求能同时整除 $m$ 和 $n$ 的最大正整数。但是要在计算机中实现的话，仅有数学的思维是不行的，计算机中实现的步骤如下：

（1）以 $n$ 除 $m$ 并令所得余数为 $r$，$r$ 必小于 $n$；

（2）若 $r=0$ 算法结束，输出结果 $n$。否则继续步骤（3）；

（3）将 $m$ 置换为 $n$，$n$ 置换为 $r$，并返回步骤（1）继续进行。

### 2.4.2　算法的性质

著名计算机科学家 Donald Knuth 在他的著作 *The Art of Computer Programming* 曾把算法的性质归纳为以下五点：

（1）输入：一个算法必须有零个或以上输入量。

（2）输出：一个算法应有一个或以上输出量，输出量是算法计算的结果。

（3）明确性：算法的描述必须无歧义，以保证算法的实际执行结果是精确地符合要求或期望，通常要求实际运行结果是确定的。

（4）有限性：依据图灵的定义，一个算法是能够被任何图灵完备系统模拟的一串运算，而图灵机只有有限个状态、有限个输入符号和有限个转移函数（指令）。而一些定义更规定算法必须在有限个步骤内完成任务。

（5）有效性：又称可行性。算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现。

### 2.4.3 算法的描述

算法是对解题过程的精确描述。定义解决问题的算法对程序员来说通常是最具挑战性的任务。它既是一种技能又是一门艺术，要求程序员懂得程序设计概念并具有创造性。对算法的描述是建立在语言基础之上的。在将算法转化为高级语言源程序之前，通常先采用文字或图形工具来描述算法。文字工具如自然语言、伪代码等，图形工具如流程图、N-S流程图等。

#### 1. 算法描述方法

1）自然语言

自然语言即人们日常生活中所用的语言，如汉语、英语等。使用自然语言不用专门训练，所描述的算法也通俗易懂。然而其缺点也是明显的：首先是由于自然语言的歧义性容易导致算法执行的不确定性；其次是由于自然语言表示的串行性，因此当一个算法中循环和分支较多时就很难清晰地表示出来；此外，自然语言表示的算法不便转换成用计算机程序设计语言表示的程序。

2）流程图

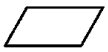
流程图是采用一些框图符号来描述算法的逻辑结构，每个框图符号表示不同性质的操作。流程图可以很方便地表示任何程序的逻辑结构。另外，用流程图表示的算法不依赖于任何具体的计算机和程序设计语言，从而有利于不同环境的程序设计。早在20世纪60年代，美国国家标准协会ANSI（American National Standards Institute）就颁布了流程图的标准，这些标准规定了用来表示程序中各种操作的流程图符号，例如用矩形表示处理，用菱形表示判断，用平行四边形表示输入/输出，用带箭头的折线表示流程等等。如表2-2所示为流程图符号及意义。

表2-2 流程图常用符号

| 流程图符号 | 名 称 | 说 明 |
| --- | --- | --- |
|  | 起止框 | 表示算法的开始和结束 |
|  | 处理框 | 表示完成某种操作，如初始化或运算赋值等 |
|  | 判断框 | 表示根据一个条件成立与否，决定执行两种不同操作的其中 |

| | | |
|---|---|---|
| | | 一个 |

续表

| 流程图符号 | 名　称 | 说　明 |
|---|---|---|
|  | 输入输出框 | 表示数据的输入输出操作 |
|  | 流程线 | 用箭头表示程序执行的流向 |
|  | 连接点 | 用于流程分支的连接 |

3）N-S 流程图

N-S 流程图又称为结构化流程图，于 1973 年由美国学者 I. Nassi 和 B. Shnei-derman 提出。与传统流程图不同的是，N-S 流程图不用带箭头的流程线来表示程序流程的方向，而采用一系列矩形框来表示各种操作，全部算法写在一个大的矩形框内，在大框内还可以包含其他从属于它的小框，这些框一个接一个从上向下排列，程序流程的方向总是从上向下。N-S 结构化流程图比较适合于表达三种基本结构（顺序、选择、循环），适于结构化程序设计，因此很受程序员欢迎。

N-S 流程图用以下的流程图符号表示：

顺序结构。顺序结构如图 2-14 所示。A 和 B 两个框组成一个顺序结构。

选择结构。选择结构如图 2-15 所示。它与图 2-14 相对应。当 p 条件成立时执行 A 操作，p 不成立则执行 B 操作。注意：图 2-15 是一个整体，代表一个基本结构。

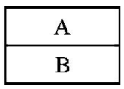



图 2-14

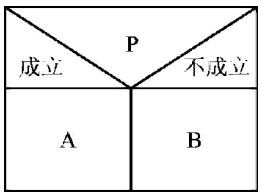
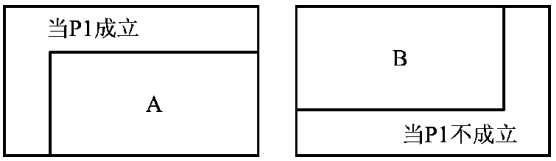



图 2-15




（a）　　（b）

图 2-16

循环结构。“当”型循环结构如图 2-16（a）所示。图 2-16（b）表示当 p1 条件成立时反复执行 A 操作，直到 p1 条件不成立为止。“直到”型循环结构用图 2-16（b）形式表示。

在初学时，为清楚起见，可如图 2-16（a）和图 2-16（b）那样，写明“当 p1”或“直到 p2”，待熟练之后，可以不写“当”和“直到”字样，只写“p1”和“p2”。从图的形状即可知道是“当”型或“直到”型。

用以上三种 N-S 流程图中的基本框可以组成复杂的 N-S 流程图，以表示算法。

应当说明，在图中的 A 框或 B 框，可以是一个简单的操作（如读入数据或打印输出等），也可以是 3 个基本结构之一。

例如图 2-17 所示表示的顺序结构，其中的 A 框可以又是一个选择结构，B 框可以又是一个循环结构，由 A 和 B 这两个基本结构组成一个顺序结构。如图 2-18 所示的直到型循环结构也是构成顺序结构中的一个子块。

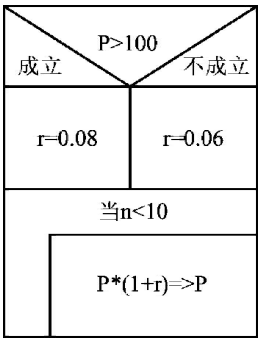



图 2-17


图 2-18

4）伪代码

伪代码是指不能够直接编译运行的程序代码，是用介于自然语言和计算机语言之间的文字和符号来描述算法和进行语法结构讲解的一个工具。表面上它很像高级语言的代码，但又不像高级语言那样要接受严格的语法检查。它比真正的程序代码更简明，更贴近自然语言。它不用图形符号，因此书写方便，格式紧凑，易于理解，便于向计算机程序设计语言算法程序过渡。用伪代码书写算法时，既可以采用英文字母或单词，也可以采用汉字，以便于书写和阅读。它没有固定的、严格的语法规则，只要把意思表达清楚即可。用伪代码描述算法时，自上而下地往下写。每一行（或每几行）表示一个基本操作。用伪代码书写的算法格式紧凑，易于理解，便于转化为计算机语言算法（即程序）。在书写时，伪代码采用缩进格式来表示三种基本结构。一个模块的开始语句和结束语句都靠着左边界书写，模块内的语句向内部缩进一段距离，选择结构和循环结构内的语句再向内缩进一段距离。这样的话，算法书写格式一致，富有层次，清晰易读，能直观地区别出控制结构的开始和结束。

5）程序设计语言

对一些简单的问题，可以直接使用某种程序设计语言来描述算法。

## 2. 算法设计举例

【例 2-3】　若给定两个正整数 *m* 和 *n*，试写出求它们的最大公约数（即能同时整除 *m* 和 *n* 的最大正整数）的算法。

解：在公元前 300 年左右，欧几里得在其著作《几何原本》(Elements)中阐述了求解两个数最大公约数的过程。

第 1 步：读入两个正整数 $m$ 和 $n$，大的数存入 $m$，小的数存入 $n$；

第 2 步：求 $m$ 除以 $n$ 的余数 $r$；

第 3 步：用 $n$ 的值取代 $m$，$r$ 的值取代 $n$；

第 4 步：判别 $r$ 的值是否为零，如果 $r = 0$，则算法结束，输出 $m$ 的值，即为最大公因子。否则返回第 2 步；

流程图如图 2-19 所示：

N-S 流程图如图 2-20 所示：

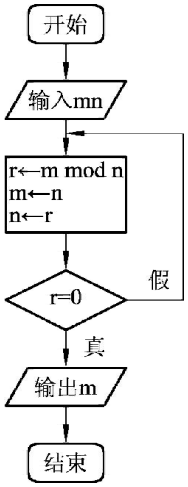



图 2-19　流程图


图 2-20　N-S 流程图

伪代码描述表示如下：

```
INPUT m,n
DO
    r=m MOD n
    m=n
    n=r
LOOP UNTIL r=0
PRINT m
END
```

C 语言描述如下：

```
#include <stdio.h>
int main( )
{
    int m,n,r ;
    printf("请输入 m,n:") ;
    scanf("%d%d",&m,&n) ;
```

```
    do
     {
          r = m%n ;
          m = n ;
          n = r ;
     }while(r) ;
     printf("最大公约数：%d\n",m) ;
     return 0 ;
}
```

【例 2-4】　求 1+2+3+…+100 之和。分别用传统流程图、N-S 流程图及自然语言描述其算法，并将该算法转化为 C 语言源程序。设变量 x 表示被加数，y 表示加数。

解：

该算法用自然语言描述如下：

步骤 1：将 1 赋值给 x；

步骤 2：将 2 赋值给 y；

步骤 3：将 x 与 y 相加，结果存放在 x 中；

步骤 4：将 y 加 1 结果存放在 y 中；

步骤 5：若 y 小于或等于 100 转到步骤 3 继续执行，否则算法结束，结果为 x。

流程图如图 2-21 所示，N-S 流程图如图 2-22 所示。

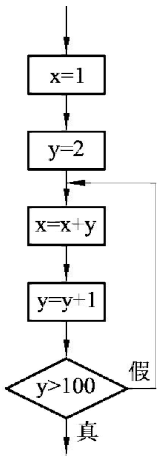



图 2-21　流程图

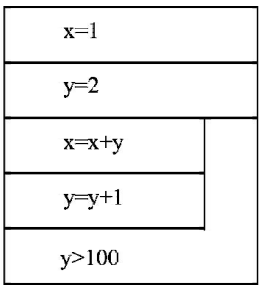



图 2-22　N-S 流程图

伪代码描述表示如下：

```
x = 1
y = 2
```

```
WHILE y< = 100
    x = x+y
    y = y+1
WEND
print ″x = ″；x
END
```

C 语言描述如下：

```
#include <stdio.h>
int main( )
{
    int x，y；
    x=1；
    y=2；
    do
    {
        x = x+y；
        y=y+1；
    }while(y<=100)；
    printf("1+2+3+...+100=%d\n"，x)；
}
```

【例 2-5】 描述商家给客户打折问题，规定一种商品一次消费金额超过 200 元的客户可以获得折扣（10%)。

解：

该算法用自然语言描述如下：

步骤 1：输入 price，qyt；

步骤 2：price 和 qyt 相乘赋给 sum；

步骤 3：判断 sum 是否大于 200，如果大于转到步骤 4，否则转到步骤 5；

步骤 4：sum 乘以 0.1 赋值给 discount，sum 减去 discount 赋给 rsum 结束；

步骤 5：sum 赋给 rsum 结束。

流程图如图 2-23 所示：

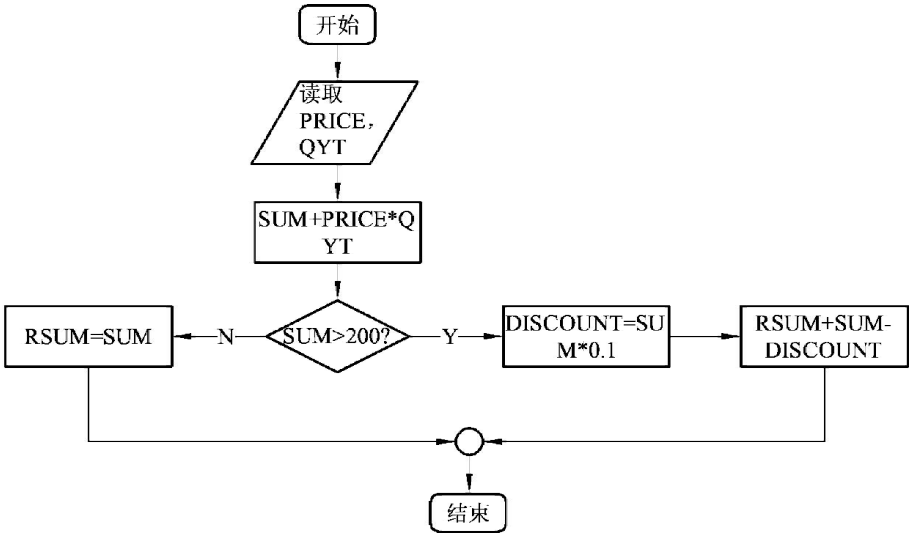



图 2-23　传统流程图

N-S 流程图如图 2-24 所示。

伪代码描述表示如下：

```
sum = qyt*price
if   (sum > 200)
      discount = sum*0.1
```

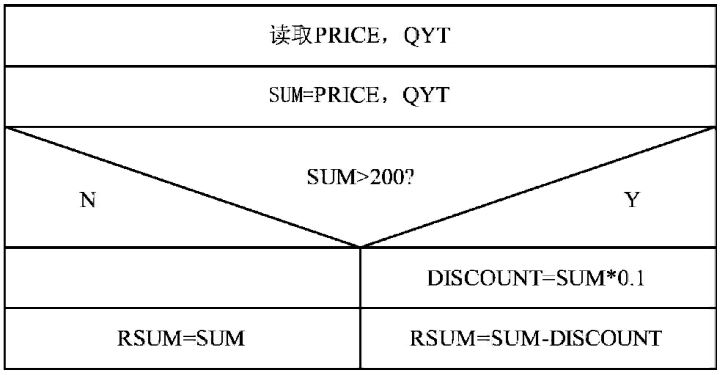



图 2-24　N-S 流程图

```
      rsum = sum - discount
else
      rsum = sum
```

C 语言描述如下：

```
#include <stdio.h>
int main( )
{
    int price，qyt，sum，discount，rsum；
```

```
    printf("请输入 price，qyt：")；
    scanf("%d%d"，&price，&qyt)；
    sum = price *qyt；
    if(sum > 200)
    {
        discount = sum*0.1；
        rsum = sum - discount；
    }
    else
     {
        rsum = sum；
     }
     printf("sum=%d，rsum=%d\n"，sum，rsum)；
}
```

## 2.5 常用算法介绍

算法包括数值计算算法和非数值计算算法两类，后者尤其重要。关于算法的概念及应用的较深入的内容将在数据结构等课程中学习，在此只对一些典型的简单的算法做概要介绍。

### 2.5.1 递 归

递归是指一个特殊的过程，在该过程中用自身的简单情况来定义自身，再自己调用自己，则称该过程是递归的。

递归是一种强有力的数学工具，它可使问题本身的描述和求解变得简洁和清晰。递归算法常常比非递归算法更容易设计，尤其当问题本身或所涉及的数据结构是递归定义的时候，使用递归算法特别适合。

递归在计算机系统内部是用栈来实现的。每一次调用过程，系统就为本次调用的参数 $n$ 开辟一个新的存储单元。递归前进到哪一层，哪一层的变量就起作用。当前递归段结束，返回上一递归调用层时，就释放掉低层的同名变量。在调用过程中，逐步深入，再逐步返回。

例如，非负整数 $n$ 的阶乘是这样定义的，即 $n=0$ 时 $n!=1$，否则 $n!=n\times(n-1)\ldots\times1$。若采用递归形式，该问题可用如下形式解决：

$$n!=\begin{cases}1 & (n=0)\\ n\cdot(n-1)! & (n>0)\end{cases}$$

若设函数 $f(n)=n!$，则该函数的递归定义为：

$$f(n)=\begin{cases}1 & (n=0)\\ n*f(n-1) & (n>0)\end{cases}$$

用 BASIC 语言表示该算法的递归程序如下：

```
FUNCTION fact(n)
IF n = 0 THEN
        fact = 1
ELSE
        fact = fact(n-1) *n
END IF
END FUNCTION
```

又如汉诺塔问题，法国数学家爱德华·卢卡斯曾编写过一个印度的古老传说：在世界中心贝拿勒斯（在印度北部）的圣庙里，一块黄铜板上插着三根宝石针。印度教的主神梵天在创造世界的时候，在其中一根针上从下到上穿好了由大到小的 64 片金片，这就是所谓的汉诺塔。不论白天黑夜，总有一个僧侣在按照下面的法则移动这些金片：一次只移动一片，不管在哪根针上，小片必须在大片上面。僧侣们预言，当所有的金片都从梵天穿好的那根针上移到另外一根针上时，世界就将在一声霹雳中消灭，而梵塔、庙宇和众生也都将同归于尽。

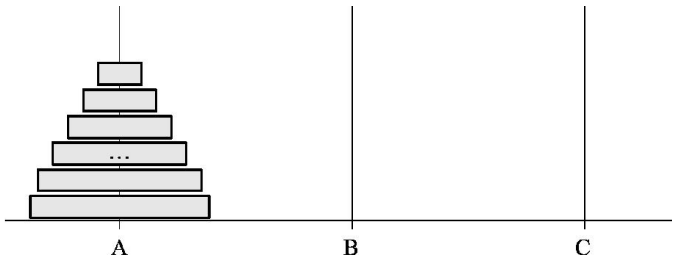



图 2-25　汉诺塔

不管这个传说的可信度有多大，如果考虑一下把 64 片金片，由一根针上移到另一根针上，并且始终保持上小下大的顺序。这需要多少次移动呢?这里需要递归的方法。假设有 *n* 片，移动次数是 *f*(*n*)，显然 *f*(1) =1，*f*(2) =3，*f*(3) =7，且 *f*(*k*+1) =2**f*(*k*) +1。此后不难证明 *f*(*n*) =2^*n* - 1。*n*=64 时，假如每秒钟一次，共需多长时间呢？一个平年 365 天有 31 536 000 秒，闰年 366 天有 31 622 400 秒，平均每年 31 556 952 秒，计算一下：18 446 744 073 709 551 615 秒。具体程序见第 8 章。

这表明移完这些金片需要 5 845.54 亿年以上，而地球存在至今不过 45 亿年，太阳系的预期寿命据说也就是数百亿年。真的过了 5 845.54 亿年，不说太阳系和银河系，至少地球上的一切生命，连同梵塔、庙宇等，都早已经灰飞烟灭。

递归算法能使一个蕴涵递归关系且结构复杂的程序简洁精练，增加可读性。递归算法在调用自身时，包含了再次调用自身，因此过程的递归调用是无休止地调用自身，但在程序中要避免这种情况的发生，所以通常是有条件地递归调用，否则递归永远也不会结束，例如上

例中就设立了一个当 $n=0$ 时的条件来结束递归过程。

### 2.5.2 枚举法

枚举法就是列举所有可能出现的情况，分别判断并把满足条件的选择出来。

例如，若公鸡（$x$）每只 3 元，母鸡（$y$）每只 5 元，小鸡（$z$）每 3 只一元，求用 100 元买 100 只鸡有多少种方案。可列出如下的联立方程：

$$\begin{cases} x+y+z=100 \\ 3x+5y+z/3=100 \end{cases}$$

两个方程式不可能求 3 个未知数，但利用计算机的高速运算特点可以给定 $x$、$y$、$z$ 的各种组合值来试算，只要结果满足两个方程，就得到并记录该方案。但是，当枚举次数超过 $10^{10}$ 时实用性不大。

### 2.5.3 查 找

查找是一种在列表中确定目标所在的位置的算法。在列表中，查找意味给定一个值，并在包含该值的列表中找到具有该值的第一个元素的位置（索引）。

例如，在图 2-26 中查找值为 32 的元素，返回的值是该元素在列表中的位置（为 3）。

| 12 | 45 | 32 | 0 | 67 | 4 | 21 | 73 | 1 | 59 |
|---|---|---|---|---|---|---|---|---|---|

图 2-26 查找

对于列表有两种基本的查找方法：顺序查找和折半查找，顺序查找可以在任何列表中查找，折半查找则要求列表是有序的。

顺序查找是从表头开始查找的，当找到目标元素或确信查找目标不在列表中时，查找过程结束，顺序查找算法的效率较低。如果列表是有序的，可以使用效率更高的折半查找算法进行查找。折半查找是从测试列表的中间元素开始查找的，如果目标元素在前半部分，就不需查找后半部分了，反之亦然。这样通过判断可以排除一半的列表元素，重复这个过程直至找到目标或是目标不在该列表中。