第一章

数据处理概述

1.1 数学预备知识

1.1.1 集合及其运算

(一) 集合的概念

所谓集合，是指若干个或无穷多个具有相同属性的元(元素)的集体。

通常，一个集合名称用大写字母表示，而集合中的某个元素用小写字母表示。

如果集合 M 由 $n(n\geqslant 0)$个元素 $a_1,a_2,\cdots,a_n$ 组成，则称集合 M 为有限集。例如，大于 1 而小于 100 的所有整数构成的集合 A 为有限集。如果一个集合中有无穷多个元素，则称此集合为无限集。例如，所有整数构成的集合 Z，所有实数构成的集合 R，大于 0 而小于 1 的所有实数构成的集合 B 等均为无限集。不包含任何元素的集合称为空集。例如，大于 1 而小于 2 的整数构成的集合为空集。空集通常用∅表示。

如果 M 是一个集合，a 是集合 M 中的一个元素，则记作 $a\in M$，称元素 a 属于集合 M；如果 a 不是集合 M 中的元素，则记作 $a\notin M$，称元素 a 不属于集合 M。

对于一个集合，通常用以下两种方法表示。

(1) 列举法

用列举法表示一个集合是将此集合中的元素全部列出来，或者列出若干项但能根据规律知其所有的元素。例如，上述举出的几个集合的例子可以用如下列举法表示出来：

大于 1 而小于 100 的所有整数的集合 A 表示为

$$A=\{2,3,4,\cdots,99\},\text{有限集}$$

所有整数构成的集合 Z 表示为

$$Z=\{0,\pm 1,\pm 2,\pm 3,\cdots\},\text{无限集}$$

空集表示为

$$\varnothing=\{\},\text{空集}$$

(2) 性质叙述法

用性质叙述法表示一个集合是将集合中的元素所具有的属性描述出来。例如：

大于 1 而小于 100 的所有整数的集合 A 表示为

$$A=\{a\mid 1<a<100\text{ 的所有整数}\}$$

所有整数构成的集合 Z 表示为

$$Z=\{z\mid z\text{ 为一切整数}\}$$

大于 0 而小于 1 的所有实数构成的集合 B 表示为

$$B=\{b\mid 0<b<1\text{ 的所有实数}\}$$

所有实数构成的集合 R 表示为

$$R=\{r|r\text{ 为一切实数}\}$$

设 M 与 N 为两个集合。如果集合 M 中的每一个元素也都为集合 N 的元素，则称集合 M 为 N 的子集，记作 $M\subseteq N$ 或 $N\supseteq M$。如果 $M\subseteq N$，且 N 中至少有一个元素 $a\notin M$，则称 M 是 N 的真子集，记作 $M\subset N$ 或 $N\supset M$。如果 $M\subseteq N$ 且 $N\subseteq M$，则称集合 M 和集合 N 相等，记作 $M=N$。

(二) 集合的基本运算

(1) 两个集合的并(union)

设有两个集合 M 和 N，它们的并集记作 $M\cup N$，其定义如下：

$$M\cup N=\{\alpha|\alpha\in M\text{ 或 }\alpha\in N\}$$

两个集合 M 与 N 的并集是指 M 与 N 中所有元素(去掉重复的元素)组成的集合。

(2) 两个集合的交(intersection)

设有两个集合 M 和 N，它们的交集记作 $M\cap N$，其定义如下：

$$M\cap N=\{\alpha|\alpha\in M\text{ 且 }\alpha\in N\}$$

两个集合 M 与 N 的交集是指 M 与 N 中所有共同元素组成的集合。

两个集合 M 与 N 的并、交均满足交换律，即

$$M\cup N=N\cup M,\ M\cap N=N\cap M$$

(3) 两个集合的差(difference)

设有两个集合 M 和 N，M 和 N 的差集记作 $M-N$，其定义如下：

$$M-N=\{\alpha|\alpha\in M\text{ 但 }\alpha\notin N\}$$

两个集合的差不满足交换律，即

$$M-N\neq N-M$$

例 1.1 设集合

$$A=\{a,b,c,d,e\}$$
$$B=\{d,e,f,g,h\}$$

则

$$A\cup B=\{a,b,c,d,e,f,g,h\}$$
$$A\cap B=\{d,e\}$$
$$A-B=\{a,b,c\}$$
$$B-A=\{f,g,h\}$$

对于集合的并、交、差有以下几个基本性质：

(1) 结合律

$$(A\cap B)\cap C=A\cap(B\cap C)$$
$$(A\cup B)\cup C=A\cup(B\cup C)$$

(2) 分配律

$$A\cap(B\cup C)=(A\cap B)\cup(A\cap C)$$
$$A\cup(B\cap C)=(A\cup B)\cap(A\cup C)$$

(3) $(A-B)\cup(B-A)=(A\cup B)-(A\cap B)$

(4) $B\cap(A-B)=\varnothing$

$$(A\cap B)\cup(A-B)=A$$

（三）映射

定义 1.1 设 A、B 是两个非空集。如果根据一定的法则 f，对于每一个 $x\in A$，在 B 中都有唯一确定的元素 y 与之对应，则称 f 是定义在 A 上而在 B 中取值的映射，记作 $f:A\to B$。并将 x 与 y 的关系记作 $y=f(x)$。x 称为自变元，y 称为在 f 作用下 x 的像。

集合 A 称为 f 的定义域，$f(A)=\{f(x)\mid x\in A\}$ 称为 f 的值域。

定义 1.2 设给定映射 $f:A\to B$，且 $B=f(A)$（即 f 的像充满整个 B）。如果对于每个 $y\in B$，仅有唯一的 $x\in A$ 使 $f(x)=y$，则称 f 有逆映射 f^{-1}（它是定义在 f(A) 上而取值于 A 的映射）。当映射 $f:A\to f(A)$ 有逆映射时，则称 f 是一一映射。

定义 1.3 若 A、B 两集合有一一映射 f 存在，使 $f(A)=B$，则称 A 与 B 成一一对应。

如果集合 A 与 B 为一一对应，则称它们互相对等，并记作 $A\sim B$。当两个集合互相对等时，称它们有相等的浓度（或元素个数）。

例 1.2 设两集合为

$$A=\{1,2,\cdots,10\}=\{x\mid 1\leqslant x\leqslant 10 \text{ 的整数}\}$$

$$B=\{1,2,\cdots,20\}=\{y\mid 1\leqslant y\leqslant 20 \text{ 的整数}\}$$

若映射 $f:A\to B$ 为

$$y=2x$$

其中定义域为 A，值域为

$$f(A)=\{2,4,6,\cdots,20\}$$

显然，映射 f 不是一一映射。

例 1.3 设两集合为

$$A=\{x\mid 0\leqslant x\leqslant 4 \text{ 的所有实数}\}$$

$$B=\{y\mid 0\leqslant y\leqslant 2 \text{ 的所有实数}\}$$

考虑映射 $f:A\to B$ 为

$$y=\sqrt{x}$$

其中定义域为 A，值域为 $f(A)=B$。

显然，f 为一一映射，集合 A 与 B 互相对等，即 $A\sim B$。

集合的对等满足以下性质：

(1) 自反性，即 $A\sim A$；

(2) 对称性，即若 $A\sim B$，则 $B\sim A$；

(3) 传递性，即若 $A\sim B$ 且 $B\sim C$，则 $A\sim C$。

1.1.2 自然数集与数学归纳法

由所有自然数所组成的集合

$$\{1,2,3,\cdots\}$$

称为自然数集。自然数集是一个无限集。

由自然数组成的集合均是自然数集的子集。自然数集的子集可以是有限集，也可以是无限集。

与自然数集对等（即具有相等浓度）的集合称为可列集（或可数集）。任一可列集中的元素排列时可标以正整数下标，即任意可列集 M 均可写成

$$M=\{a_1,a_2,\cdots,a_n,\cdots\}$$

关于自然数集及其子集有以下两个命题成立。

定理 1.1 在自然数集的任一非空子集 M 中，必定有一个最小数。即在集合 M 中有不大于其它任意数的数。

证明：因为 M 非空，所以在 M 中可取得一自然数 n。

显然，M 中所有不大于 n 的自然数形成的非空集 N 包含在 M 中，即 $N\subset M$。如果 N 中有最小数，则此最小数就是 M 的最小数。

而在 N 中最多有 n 个自然数(1 到 n)，因此 N 中有一个最小数。

综上所述，在自然数集的任一非空子集 M 中必定有一个最小数。定理得证。

定理 1.2 假定 M 是由自然数形成的集合，如果它含有 $1,2,\cdots,k$，并且当它含有数 $n-1,n-2,\cdots,n-k(n>k)$ 时，也含有数 n，那么它含有所有的自然数，即 M 是自然数集。

证明：设 N 是所有不属于 M 的自然数形成的集合，则 $1,2,\cdots,k\notin N$。

现假设 N 不是空集，则由定理 1.1 可知：在 N 中必定有一个最小数。设此最小数为 c。

由于 c 是 N 中的最小数，即 $c\in N$，因此 $c\notin M$，且 $c\neq 1,2,\cdots,k$，同时，$c-1,c-2,\cdots,c-k$ 均为自然数。又由于 c 是 N 中的最小数，所以自然数 $c-1,c-2,\cdots,c-k\notin N$，即 $c-1,c-2,\cdots,c-k\in M$，而根据定理中的条件有 $c\in M$。

由上所述，一方面有 $c\notin M$，另一方面又有 $c\in M$，这就导致矛盾。这个矛盾是由于一开始假定 N 不是空集所造成的。因此 N 只能为空集，即所有自然数均在 M 中，M 为自然数集。

定理得证。

上述定理是数学归纳法的基础。通常，为了证明一个命题对于所有的自然数是真，采用数学归纳法证明的步骤如下：

(1) 证明命题对于自然数 $1,2,\cdots,k$ 是真的；

(2) 假设命题对于自然数 $n-k,n-k+1,\cdots,n-1(n>k)$ 是真的(这一步称为归纳假设)；

(3) 证明命题对于自然数 n 也是真的。

上述步骤(1)中 k 值的选取决定于在步骤(3)的证明过程中要用到归纳假设的最小自然数。在步骤(3)中，为了证明命题对于自然数 n 是真，要用到归纳假设的最小自然数如果为 $n-k$，则在步骤(1)中要对 1 到 k 中的所有自然数证明命题为真。

例 1.4 证明任意一笔大于 7 元的整数付款均可用 3 元及 5 元的票面的钞票支付。

这个问题相当于要证明下列命题：

对于任意的自然数 n，存在一对非负整数(i,j)有

$$7+n=3i+5j$$

下面用数学归纳法证明这个命题。

证明：

(1) 当 $n=1$ 时，有 $7+1=3+5$，即 $i=1,j=1$，命题成立。

当 $n=2$ 时，有 $7+2=3\times 3+5\times 0$，即 $i=3,j=0$，命题成立。

当 $n=3$ 时，$7+3=3\times 0+5\times 2$，即 $i=0,j=2$，命题成立。

(2) 假设命题对于自然数 $n-1,n-2,n-3(n>3)$ 成立(归纳假设)。

(3) 考虑自然数 n，有

$$7+n=[7+(n-3)]+3$$

根据归纳假设，对于自然数 $n-3$ 命题成立，设存在一对非负整数 i_1, j_1 有

$$7+(n-3)=3i_1+5j_1$$

则有

$$\begin{aligned}7+n &=[7+(n-3)]+3=3(i_1+1)+5j_1\\ &=3i+5j\end{aligned}$$

其中 $i=i_1+1, j=j_1$ 均为非负整数。即对于自然数 n 命题也成立。

由此得出结论，对于所有的自然数 n 命题成立。

在这个例子中，由于步骤(3)的证明过程中要用到归纳假设的最小自然数为 $n-3$，因此在步骤(1)中取 $k=3$。

1.1.3 笛卡尔积

在 1.1.1 中介绍了两个集合的并、交、差运算。对于集合，还有一种很重要的运算，即笛卡尔积(Cartesian Product)。

设有 n 个集合 $D_1, D_2, \cdots, D_n$，此 n 个集合的笛卡尔积定义为

$$D_1\times D_2\times\cdots\times D_n=\{(d_1,d_2,\cdots,d_n)\mid d_i\in D_i, i=1,2,\cdots,n\}$$

其中 $(d_1,d_2,\cdots,d_n)$ 称为 n 元组(n-tuple)，d_i 称为 n 元组的第 i 个分量。

由笛卡尔积的定义可以看出，n 个集合的笛卡尔积是以 n 元组为元素的集合，而每一个 n 元组中的第 i 个分量取自于第 i 个集合 D_i。

例 1.5 设有三个集合

$$A=\{a_1,a_2,a_3\}, B=\{b_1,b_2\}, C=\{c_1,c_2\}$$

则它们的笛卡尔积为

$$\begin{aligned}A\times B\times C=\{&(a_1,b_1,c_1),(a_1,b_1,c_2),(a_1,b_2,c_1),(a_1,b_2,c_2),(a_2,b_1,c_1),(a_2,b_1,c_2),\\ &(a_2,b_2,c_1),(a_2,b_2,c_2),(a_3,b_1,c_1),(a_3,b_1,c_2),(a_3,b_2,c_1),(a_3,b_2,c_2)\}\end{aligned}$$

如果 n 个集合 $D_1, D_2, \cdots, D_n$ 中的元素个数分别为 $m_1, m_2, \cdots, m_n$，则其笛卡尔积中共有 $m_1\times m_2\times\cdots\times m_n$ 个 n 元组。即 n 个集合的笛卡尔积是所有 n 元组组成的集合。

1.1.4 二元关系

定义 1.4 设 M 和 N 是两个集合，则其笛卡尔积

$$M\times N=\{(x,y)\mid x\in M \text{ 且 } y\in N\}$$

的每一个子集称为在 $M\times N$ 上的一个二元关系。

如果 $M=N$，则其笛卡尔积

$$M\times N=\{(x,y)\mid x,y\in M\}$$

的每一个子集称为在集合 M 上的一个二元关系，简称为在集合 M 上的一个关系。

例 1.6 设集合 M 为

$$M=\{a,b,c,d,e,f\}$$

则下列各二元组的集合为在集合 M 上的一个关系：

$$R_1=\{(a,b),(b,c),(c,d),(d,e),(e,f)\}$$

$$R_2=\{(a,e),(a,a),(c,f),(d,b),(e,a),(f,c),(b,d)\}$$

$$R_3 = \{(a,a),(b,b),(c,c),(d,d),(e,e),(f,f),(c,f),(e,a)\}$$

$$R_4 = \{(a,b),(b,e),(c,d),(d,f),(a,e),(c,f)\}$$

集合 M 上的一个关系实际上反映了 M 中各元素之间的联系。

定义 1.5 设 R 是集合 M 上的一个关系。

(1) 如果(a,b)∈R,则称 a 是 b 的关于 R 的前件(predecessor),b 是 a 的关于 R 的后件(successor)。

(2) 如果对于每一个 a∈M,都有(a,a)∈R,则称关系 R 是自反的(reflexive);如果对于任何 a∈M,(a,a)∈R 均不成立,则称关系 R 是非自反的(antireflexive)。

(3) 如果(a,b)∈R 时必有(b,a)∈R,则称关系 R 是对称的(symmetric)。

(4) 如果当(a,b)∈R 且(b,c)∈R 时必有(a,c)∈R,则称关系 R 是传递的(transitive)。

在例 1.6 中,关系 R_1 是非自反的,但不是对称的,也不是传递的;关系 R_2 是对称的,但不是自反的,也不是非自反的,也不是传递的;关系 R_3 是自反的,但不是对称的,也不是传递的;关系 R_4 是传递的,且是非自反的,但不是对称的。

由此可以看出,集合 M 中的各元素之间的逻辑关系可以由集合 M 上的一个关系来描述。

定义 1.6 设 R 是 M 上的一个传递关系,且 T⊆R。若对于任何(x,y)∈R,在 M 中有元素 $x_0,x_1,x_2,\cdots,x_n$($n\geqslant 1$)满足:(1) $x_0=x$,(2) $x_n=y$,(3) $(x_{i-1},x_i)\in T(i=1,2,\cdots,n)$。则称关系 T 是关系 R 的基(basis),又称关系 R 是关系 T 的传递体(transitive hull)。

例 1.7 设集合 M 为

$$M = \{1,2,3,4,5\}$$

集合 M 上的一个关系为

$$R = \{(1,2),(2,3),(3,5),(3,4),(1,3),(1,4),(2,4),(2,5),(1,5)\}$$

可以验证,关系 R 是非自反的,且是传递的。现考虑集合 M 上的另一个关系

$$T = \{(1,2),(2,3),(3,5),(3,4)\}$$

由于 T⊆R,且对于关系 R 中的每一个二元组(x,y),在 M 中存在元素 $x_0,x_1,\cdots,x_n$,满足定义 1.6 中的三个条件。验证过程如下:

R	$x_0,x_1,\cdots,x_n$	
(1,2)	1,2	(1,2)∈T
(2,3)	2,3	(2,3)∈T
(3,5)	3,5	(3,5)∈T
(3,4)	3,4	(3,4)∈T
(1,3)	1,2,3	(1,2),(2,3)∈T
(1,4)	1,2,3,4	(1,2),(2,3),(3,4)∈T
(2,4)	2,3,4	(2,3),(3,4)∈T
(2,5)	2,3,5	(2,3),(3,5)∈T
(1,5)	1,2,3,5	(1,2)(2,3),(3,5)∈T

因此,T 是 R 的具有 4 个元素(即二元组)的基。

同样还可以验证关系

$$T_1 = \{(1,2),(1,4),(2,3),(3,5),(3,4)\}$$

是R的具有5个元素的基。但关系

$$T_2 = \{(1,2),(2,3),(3,4),(4,5)\}$$

不是R的基，因为$(4,5) \notin R$。

1.2 算　法

计算机科学的发展，为科学计算及数据处理提供了高速和高精度的计算工具。但计算机只能机械地执行人的指示和命令，它不会主动地进行思维，不可能发挥任何创造性。因此，用计算机解决一个实际问题，首先要进行程序设计。通常，程序设计主要包括两个方面：行为特性的设计与结构特性的设计。行为特性的设计要求将解决实际问题的每个细节准确地加以定义，并且还应当将全部解题过程完整地描述出来，这一过程即是算法的设计。结构特性的设计是指确定合适的数据结构。数据结构与算法之间有着密切的关系。特别是对于数据处理问题，算法的效率通常与数据在计算机内的表示法（称为数据的存储结构）有直接的关系。在实际应用中，对于不同的数据表示形式，也将采用不同的算法。

本节主要讨论算法的设计与分析。有关数据结构的概念将在1.3节讨论。

1.2.1 算法的概念

概括地说，“算法”是指解题方案的准确而完整的描述。

对于一个问题，如果可以通过一个计算机程序，在有限的存储空间内运行有限长的时间而得到正确的结果，则称这个问题是算法可解的。但算法不等于程序，也不等于计算方法。程序可以作为算法的一种描述，但程序通常还需考虑很多与方法和分析无关的细节问题，这是因为在编写程序时要受到计算机系统运行环境的限制。通常，程序设计不可能优于算法的设计。

作为一个算法，应具有以下四个特征。

（1）能行性（effectiveness）

算法的能行性包括两个方面：一是算法中的每一个步骤必须是能实现的，例如，在算法中，不允许出现分母为零的情况，在实数范围内不能求一个负数的平方根等；二是算法执行的结果要能达到预期的目的。

针对实际问题设计的算法，人们总是希望能够得到满意的结果。算法总是与特定的计算工具有关。例如，某计算工具具有七位有效数字（如FORTRAN中的单精度运算），在计算下列三个量

$$A = 10^{12},\quad B = 1,\quad C = -10^{12}$$

的和时，如果采用不同的运算顺序，就会得到不同的结果，即

$$A + B + C = 10^{12} + 1 + (-10^{12}) = 0$$

$$A + C + B = 10^{12} + (-10^{12}) + 1 = 1$$

而在数学上，A+B+C与A+C+B是完全等价的。因此，算法与计算公式是有差别的。在设计一个算法时，必须考虑它的能行性，否则是不会得到满意结果的。

（2）确定性（definiteness）

算法的确定性，是指算法中的每一个步骤都必须是有明确定义的，不允许有模棱两可的

解释，也不允许有多义性。这一性质也反映了算法与数学公式的明显差异。在解决实际问题时，可能会出现这样的情况：针对某种特殊问题，数学公式是正确的，但按此数学公式设计的计算过程可能会使计算机系统无所适从。这是因为根据数学公式设计的计算过程只考虑了正常使用的情况，而当出现异常情况时，此计算过程就不能适应了。例如，某计算工具规定：大于 100 的数认为是比 1 大很多，而小于 10 的数不能认为是比 1 大很多；且在正常情况下出现的数或是大于 100，或是小于 10。但指令“输入一个 x，若 x 比 1 大很多，则输出数字 1，否则输出数字 0”是不确定的。这是因为，在正常的输入情况下，这一计算可以得到正确的结果，但在异常情况下（输入的 x 在 10 与 100 之间），其输出结果就不确定了。

(3) 有穷性(finiteness)

算法的有穷性是指算法必须能在有限的时间内做完，即算法必须能在执行有限个步骤之后终止。数学中的无穷级数，在实际计算时只能取有限项，即计算无穷级数值的过程只能是有穷的。因此，一个数的无穷级数表示只是一个计算公式，而根据精度要求确定的计算过程才是有穷的算法。

算法的有穷性还应包括合理的执行时间的含义。因为如果一个算法需要执行千万年，显然失去了实用价值。例如，克莱姆(Cramer)规则是求解线性代数方程组的一个数学方法，但不能以此设计为算法，这是因为，虽然总可以根据克莱姆规则设计出一个计算过程用于计算所有可能出现的行列式，但此计算过程所需的时间实际上是不能容忍的。还例如，从理论上讲，总可以写出一个正确的奕棋程序，而且这并不是一件很困难的工作。由于在一个棋盘上安排棋子的方式总是有限的，而且，根据一定的规则，在有限次地移动棋子之后比赛一定结束。因此，奕棋程序可以考虑计算机每一次可能的移动，它的对手每一次可能的应答，以及计算机对这些移动的可能应答等等，直到每个可能的移动停止下来为止。此外，由于计算机知道每次移动的最后结果，因此总可以选择一种最好的移动方式。但即使如此，这奕棋程序还是不可能执行，因为所有这些可能移动的次数太多，所要化费的时间不能容忍。由上述两个例子可以看出，虽然许多计算过程是有限的，但仍有可能无实用价值。

(4) 拥有足够的情报

一个算法是否有效，还取决于为算法所提供的情报是否足够。例如，对于指令“如果小明是学生，则输出字母 Y，否则输出字母 N”。当算法执行过程中提供了小明不是学生的情报时，执行的结果将输出字母 N；当提供的情报中只有部分学生的名单，且小明恰在其中，执行的结果将输出字母 Y。但如果在提供的部分学生的名单中，找不到小明的名字，则在执行算法过程中无法服从此指令，因为在部分学生的名单中找不到小明的名字，既不能说明小明是学生，也不能说明小明不是学生。

通常，算法中的各种运算总是要施加到各个运算对象上，而这些运算对象又可能具有某种初始状态，这是算法执行的起点或是依据。因此，一个算法执行的结果总是与输入的初始数据有关，不同的输入将会有不同的结果输出，当输入不够或输入错误时，算法本身就无法执行或执行有错。一般来说，当算法拥有足够的情报时，此算法才是有效的，而提供的情报不够时，算法并不是有效的。

综上所述，所谓算法，是一组严谨地定义运算顺序的规则，并且每一个规则都是有效的且是明确的，此顺序将在有限的次数下终止。

1.2.2 算法描述语言

算法描述语言是表示算法的工具，它只面向读者，不能直接用于计算机。在本书中，绝大部分的算法将采用本节介绍的算法描述语言来描述。在用算法语言描述一个算法时，开始一般都要对输入和输出参数作必要的说明。下面简要叙述本书采用的算法描述语言中的各个语句并加以必要的说明。

（一）符号与表达式

符号是以字母开头的字母和数字 的有限串，用以表示变量名、数组名等，必要时也用以表示语句标号。

在语句标号后应跟随一个冒号，然后是语句。例如

loop：i←i+1

在算法中，变量或数组的类型一般可从上下文看出，因此一般不需要作说明，除非在特殊情况下才作必要的说明。

有时，算法中的某些指令或子过程直接用叙述的方式给出，以便使算法更清楚。例如，“设 x 是 A 中的最大项”(其中 A 为一个数组)，“将 x 插入 L 之中”(其中 L 是某个表)，等等。

算术运算符通常用＋、－、＊、/和↑，而＊和↑在算法中一般可以省略，而沿用通常的数学表示法。

关系运算符用＝、≠、＜、＞、≤和≥来表示。

逻辑运算符用 and(与)、or(或)和 not(非)来表示。

（二）赋值语句

赋值语句的形式为

$$a \leftarrow e$$

其中 a 为变量名或数组元素，e 为算术表达式或逻辑表达式。如果 a 和 b 都为变量名或数组元素，则可用记号

$$a \rightleftharpoons b$$

表示将 a 与 b 的内容进行交换。而用记号

$$a \leftarrow b \leftarrow e$$

表示将表达式 e 的值同时赋给 a 和 b。

（三）控制转移语句

无条件转移语句用如下形式：

GOTO　标号

它导致转到具有指定标号的语句去执行。

条件转移语句有以下两种形式：

IF C THEN S

或

IF C THEN S_1

ELSE S_2

其中 C 是一个逻辑表达式，S_1 和 S_2 是单一的语句或者是用一对括号{}括起来的语句组。如

果 C 为“真”，则 S 或 S_1 被执行一次；如果 C 为“假”，在第一种形式中，IF 语句执行完成，而在第二种形式中，将执行 S_2，但在所有情况下，控制将进行到下一语句，除非在 S、S_1 或 S_2 中有 GOTO 语句使控制转到别的地方。

（四）循环语句

循环语句有两种形式：一是 WHILE 语句，二是 FOR 语句。

WHILE 语句的形式为

```
WHILE C DO S
```

其中 C 是逻辑表达式，S 是单一的语句或用一对括号{}括起来的语句组。如果 C 为“真”，则执行 S，且在每次执行 S 之后都要重新检查 C；如果 C 为“假”，控制就转到紧跟在 WHILE 语句后面的语句。由此看出，当控制第一次到达 WHILE 语句时，若 C 为“假”，则 S 一次也不执行。WHILE 语句的功能等价于如下的 IF 语句

```
loop: IF C THEN
        { S
          GOTO loop
        }
```

FOR 语句的形式为

```
FOR i=init TO limit BY step DO  S
```

其中 i 是循环控制变量，init、limit 和 step 都是算术表达式，而 S 是单一的语句或是用一对括号{}括起来的语句组。当 step＞0 时，这个语句的功能等价于如下 IF 语句：

```
  i←init
loop: IF i≤limit THEN
      { S
        i←i+step
        GOTO loop
      }
```

当 step＜0 时，这个语句的功能等价于如下 IF 语句：

```
    i←init
loop: IF i≥limit THEN
      { S
        i←i+step
        GOTO loop
      }
```

在 FOR 循环语句中，如果 step＝1，则 BY step 可以省略，变为

```
FOR i=init TO limit DO S
```

（五）其它语句

在算法描述中，还可能要用到其它一些语句，读者遇到时完全可以知道它们的含义。下面只列出几个常见的语句。

EXIT 语句通常用于退出某个循环，使控制转到包含 EXIT 语句的最内层的 WHILE 或 FOR 循环后面的一个语句。

RETURN 语句用于结束算法的执行。如果算法是在最后一行的语句之后结束，则 RE-

TURN 可以被省略。而且，在 RETURN 语句后面允许使用用引号括起来的注释信息。

READ(或 INPUT)和 OUTPUT(或 PRINT，或 WRITE)语句用于输入和输出。

最后还需说明，算法中的注释总是用一对方括号[ ]括起来。几个短语句可以写在一行上，但彼此之间用分号隔开。另外，复合语句总是用一对花括号{}括起来作为语句组。

1.2.3 算法基本设计方法

计算机解题的过程实际上是在实施某种算法，这种算法通常称为计算机算法。计算机算法不同于人工处理的算法。例如，为了计算定积分

$$S = \int_a^b f(x)dx$$

人工处理的步骤为：

(1) 找出被积函数 f(x)的源函数 F(x)；

(2) 利用牛-莱公式计算 S=F(b)--F(a)。但用计算机计算这个积分不能按照上述步骤，因为通常很难用程序来寻找被积函数的源函数，而且实际上也没有这个必要，实际问题中的被积函数往往不存在源函数。因此，在实际问题中，计算机计算定积分通常采用数值积分法，根据实际被积函数的类型及精度要求选择相应的算法。

本节主要介绍工程上常用的几个算法设计方法。有些方法之间有一定的联系。

(一) 列举法

列举法的基本思想是根据提出的问题，列举所有可能情况，并用问题中提出的条件检验哪些是需要的，哪些是不需要的。因此，列举法常用于解决“是否存在”或“有多少种可能”等类型的问题，例如求解不定方程的问题。

列举法的特点是算法比较简单，但当列举的可能情况较多时，执行列举算法的工作量将会很大。因此，在用列举法设计算法时，使方案优化，尽量减小运算工作量，是应该重点注意的。通常，只要对实际问题作详细的分析，将与问题有关的知识条理化、完备化、系统化，从中找出规律，或对所有可能的情况进行分类，引出一些有用的信息，列举量是可以减少的。

例 1.8 设每只母鸡值 3 元，每只公鸡值 2 元，两只小鸡值 1 元。现用 100 元买 100 只鸡，问能买母鸡、公鸡、小鸡各多少只？

现用列举法设计求解的算法。

最粗略的列举算法如算法 1.1。

算法 1.1 求解百鸡问题。

```
FOR I=0 TO 100 DO
FOR J=0 TO 100 DO
FOR K=0 TO 100 DO
  { M←I+J+K
    N←3I+2J+0.5K
    IF (M=100)and(N=100) THEN
        OUTPUT I,J,K
  }
RETURN
```

在算法 1.1 中，I、J、K 分别表示母鸡、公鸡、小鸡的只数。在这个算法中，内层循环需循

环 101 次，外面两层循环也各需要循环 101 次，因此总循环次数为 101^3。显然，这样大的列举量是没有必要的，只要对问题作一分析，很容易发现这个算法还可以优化，减少不必要的循环次数。

首先，考虑到母鸡为 3 元一只，因此，母鸡最多只能买 33 只，即外循环没有必要从 0 到 100，而只需要从 0 到 33 就可以了。

其次，考虑到公鸡为 2 元一只，因此，公鸡最多只能买 50 只。又考虑到对公鸡的列举是在算法的第二层循环中，且买一只母鸡的价钱相当于买 1.5 只公鸡。因此，在第一层循环中已确定买 I 只母鸡的前提下，公鸡最多只能买 50−1.5I 只，即第二层对 J 的循环只需要从 0 到 50−1.5I 就可以了。

最后，考虑到买 100 只鸡。因此，在第一层循环中已确定买 I 只母鸡，第二层已确定买 J 只公鸡的前提下，买小鸡的数量只能是 K＝100−I−J，即第三层的循环已没有必要了。

经过以上分析，可以将算法 1.1 改写成算法 1.2。

算法 1.2 求解百鸡问题。

```
FOR I=0 TO 33 DO
FOR J=0 TO 50−1.5I DO
  { K←100−I−J
    IF 3I+2J+0.5K=100 THEN
        OUTPUT I,J,K
  }
RETURN
```

算法 1.2 的列举量(即总循环次数)为

$$\sum_{I=0}^{33}(51-1.5I)\approx 894$$

列举原理是计算机应用领域中十分重要的原理。许多实际问题，若采用人工列举是不可想象的，但由于计算机的运算速度快，擅长重复操作，可以进行大量的列举。因此，列举算法是计算机算法中的一个基础算法。实际上，列举就是从某个集合中一一列举各个元素，如果为了证明一个命题为真，则只要一一列举对于每一种可能情况命题均为真就行了。

列举法是比较笨拙而原始的方法，最大的弱点是运算量大。但在有些实际问题中，局部的使用列举法却是很有效的。例如，对于寻找路径、查找、搜索等问题，局部使用列举法是一种行之有效的方法。

(二) 归纳法

列举算法具有结构简单的优点，但它有两个主要的缺点。一是对于某些实际问题，列举法的效率太低，特别是当列举量大到不能容忍时，列举法实际上就不适用了；二是在很多实际问题中，列举量为无限，此时，列举算法是无效的。因此，列举法只适用于列举量为有限的情况。

归纳法的基本思想是通过列举少量的特殊情况，经过分析，最后找出一般的关系。显然，归纳法要比列举法更能反映问题的本质。但是，从一个实际问题中总结归纳出一般的关系，并不是一件容易的事情，尤其是要归纳出一个数学模型更为困难。而且，归纳过程通常也没有一定的规则可供遵循。从本质上讲，归纳就是通过观察一些简单而特殊的情况，最后总结出有用的结论或解决问题的有效途径。通常，归纳的过程分以下四个步骤：

(1) 细心的观察；

(2) 丰富的联想；

(3) 继续尝试；

(4) 总结归纳出结论。

归纳是一种抽象，即从特殊现象中找出一般关系。但在归纳的过程中不可能对所有的情况进行列举，因而最后得到的结论还只是一种猜测(即归纳假设)。所以对于归纳假设还必须加以严格的证明。实际上，通过精心观察而提出的归纳假设得不到证实或最后证明是错的，也是常有的事。以下我们以图 1.1 所示的乌勒母方螺线为例进行分析。如果仔细观察一下就可以发现，在这个螺线形的一条对角线上的数均为素数，即这些素数为：

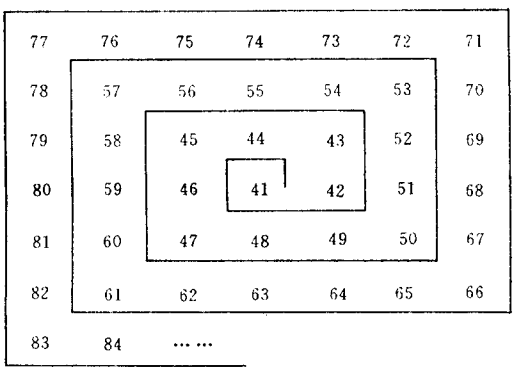

图 1.1 乌勒母方螺线

41,43,47,53,61,71,83,…

如果按照这个规律继续画下去，该对角线上的数仍为素数，即其后的素数为

97,113,131,151,…

(当然不可能无限制地画下去)。考察上面得到的素数序列可以发现：每相邻两个素数之差(后一个素数减去前一个素数)构成一个等差数列，即

$$\underbrace{41,43}_{2},\underbrace{47,53}_{4},\underbrace{61,71}_{6},\underbrace{83,97}_{8},\underbrace{113,131}_{10},\underbrace{151}_{12},\cdots$$

由此不难发现，素数序列具有如下通项公式：

$$a_n = 41 + n(n-1), n = 1,2,\cdots$$

根据这个通项公式，立刻就会发现，当 n=41 时，a_n 已经不是素数了。因此，前面的两个归纳假设(即乌勒母方螺线对角线上的数是素数与两相邻的素数之差构成等差数列)必有一个是错的。

对归纳假设的证明通常采用数学归纳法。

(三) 递推

所谓递推，是指从已知的初始条件出发，逐次推出所要求的各中间结果及最后结果。其中初始条件或是问题本身已经给定，或是通过对问题的分析与化简而确定。递推本质上也属于归纳法。工程上许多递推关系式，实际上是通过对实际问题的分析与归纳而得到的，因此，递推关系式是归纳的结果。递推算法在数值计算中是极为常见的。但对于数值型的递推算

法必须要注意数值计算的稳定性问题。

例 1.9 计算下列定积分：

$$I_n = \int_0^1 \frac{x^n}{x+5}dx, \quad n = 0,1,2,\cdots,20$$

利用牛-莱公式计算这 21 个积分显然是很复杂的。如果对这个积分稍作分析，可以得到以下关系：

$$I_n + 5I_{n-1} = \frac{1}{n} \tag{1.1}$$

根据这个关系，就可以得到如下递推公式：

$$I_n = \frac{1}{n} - 5I_{n-1} \tag{1.2}$$

即只要知道 n—1 时的积分值 I_{n-1}，就可以计算 n 时的积分值 I_n。为了确定初始条件 I_0，只要计算积分

$$I_0 = \int_0^1 \frac{1}{x+5}dx = \ln\frac{6}{5} \approx 0.182322$$

由上述的分析和归纳，可以得到计算该例 21 个积分的递推算法为

$$\begin{cases} I_0 = 0.182322 \\ I_n = \dfrac{1}{n} - 5I_{n-1}, \quad n = 1,2,\cdots,20 \end{cases} \tag{1.3}$$

其计算结果列于表 1.1 中，但只要作进一步的分析，就可以发现，由上述递推算法计算得到的结果是极不可靠的。上述积分除了具有相邻两项的关系式(1.1)以外，还满足不等式

$$0 < I_n < I_{n-1} \tag{1.4}$$

由不等式(1.4)可知，I_{20}应比 I_0 小，且所有的积分值不可能为负。而表 1.1 中的结果不符合这些要求。下面从另一种递推公式可得到正确结果。

表 1.1

n	I_n	n	I_n
0	0.182322	11	-0.216941×10^{2}
1	0.883900×10^{-1}	12	0.108554×10^{3}
2	0.580500×10^{-1}	13	-0.542692×10^{3}
3	0.430832×10^{-1}	14	0.271353×10^{4}
4	0.345842×10^{-1}	15	-0.135676×10^{5}
5	0.270790×10^{-1}	16	0.678380×10^{5}
6	0.312715×10^{-1}	17	-0.339190×10^{6}
7	-0.135005×10^{-1}	18	0.169595×10^{7}
8	0.192502	19	-0.847975×10^{7}
9	−0.851400	20	0.423988×10^{8}
10	0.435700×10^{1}		

实际上，根据关系式(1.1)还可以得到另一个递推公式

$$I_{n-1} = \frac{1}{5n} - \frac{1}{5}I_n \tag{1.5}$$

在使用递推公式(1.5)时，需要确定初始值 I_{20}。

根据不等式(1.4)有

$$5I_{n-1} < I_n + 5I_{n-1} < 6I_{n-1}$$

再根据式(1.1)，上述不等式变为

$$5I_{n-1} < \frac{1}{n} < 6I_{n-1}$$

最后可得

$$\frac{1}{6n} < I_{n-1} < \frac{1}{5n}$$

当 n=21 时，有

$$\frac{1}{6 \times 21} < I_{20} < \frac{1}{5 \times 21}$$

由于$\frac{1}{6\times21}$与$\frac{1}{5\times21}$很接近，因此可用它们的平均值作为 I_{20}的近似值，即

$$I_{20} \approx \frac{1}{2}\left(\frac{1}{6 \times 21} + \frac{1}{5 \times 21}\right) \approx 8.73016 \times 10^{-3}$$

由此可得另一个递推算法为

$$\begin{cases} I_{20} = 8.73016 \times 10^{-3} \\ I_{n-1} = \frac{1}{5n} - \frac{1}{5}I_n, \ n = 20,19,\cdots,2,1 \end{cases} \tag{1.6}$$

其计算结果列于表 1.2 中。显然，由递推算法(1.6)式计算得到的结果是正确的。

表 1.2

n	I_n	n	I_n
20	0.873016×10^{-2}	9	0.169265×10^{-1}
19	0.825397×10^{-2}	8	0.188369×10^{-1}
18	0.887552×10^{-2}	7	0.212326×10^{-1}
17	0.933601×10^{-2}	6	0.243249×10^{-1}
16	0.989751×10^{-2}	5	0.284684×10^{-1}
15	0.105206×10^{-1}	4	0.343063×10^{-1}
14	0.112292×10^{-1}	3	0.431387×10^{-1}
13	0.120399×10^{-1}	2	0.580389×10^{-1}
12	0.129766×10^{-1}	1	0.883922×10^{-1}
11	0.140713×10^{-1}	0	0.182322
10	0.153676×10^{-1}		

为什么会有这两种不同结果呢？

在递推算法(1.3)式中，初值 I_0 是近似的，只精确到小数点后第六位(受计算机的有效数字限制)，即 I_0 存在一个误差。而在用递推算法(1.3)式递推过程中，每递推计算一次，其误差就会增大到 5 倍，因此，计算到 I_{20}时，其误差是初值 I_0 误差的 5^{20}倍。

在递推算法(1.6)式中，每递推计算一次，后一个值的误差是前一个值误差的 1/5，因此，计算到 I_0 时，其误差是初值 I_{20} 误差的 $(1/5)^{20}$ 倍。有趣的是，在递推算法(1.6)式中，若将 I_{20} 取为 1 或 10 等等，最后计算得到的 I_0 仍为 0.182322。

(四) 递归

在工程实际中，有许多概念是用递归来定义的，数学中的许多函数也是用递归来定义的。递归函数在可计算性理论与算法设计中都占有很重要的地位。

描述递归定义的函数或求解递归问题的过程称为递归算法。一个递归算法，本质上是将较复杂的处理归结为较简单的处理，直到最简单的处理。因此，递归的基础也是归纳。

例 1.10 求解 Hanoi 塔问题

设有三个塔座分别为 X、Y、Z。现有 n 个直径各不相同的圆盘，且按直径从小到大用自然数编号为 1,2,…,n。开始时，此 n 个圆盘依下大上小的顺序放在塔座 X 上，如图 1.2 所示。现要根据下列原则将 X 塔座上的这 n 个圆盘移到 Z 塔座上：

(1) 每次只允许移动一个圆盘(从一个塔座到另一个塔座)；

(2) 移动时可用 Y 塔座作为中间塔座；

(3) 在移动过程中，任何一个塔座上均不允许出现大压小的情况发生。

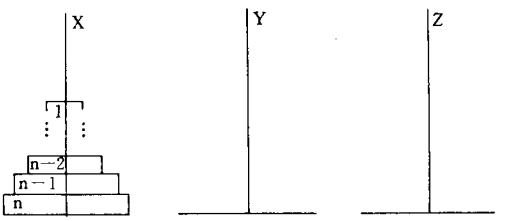

图 1.2 Hanoi 塔问题

对这个问题稍作分析，就可以得到求解的步骤：

(1) 如果 n=1，则可直接将这一个圆盘移动到目的柱上，过程结束。如果 n>1，则进行下一步。

(2) 设法将起始柱的上面 n−1 个圆盘(编号 1 至 n−1)按移动原则移到中间柱上。

(3) 将起始柱上的最后一个圆盘(编号为 n)移到目的柱上。

(4) 设法将中间柱上的 n−1 个圆盘按移动原则移到目的柱上。

在上述步骤中，(2)与(4)实际上还是 Hanoi 塔问题，只不过其规模小一些而已。如果最原始的问题为 n 阶 Hanoi 塔问题，且表示为

Hanoi(n,X,Y,Z)

则(2)与(4)为 n−1 阶 Hanoi 塔问题，分别表示为

Hanoi(n−1,X,Z,Y)

Hanoi(n−1,Y,X,Z)

其中第一个参数表示问题的阶数，第二、三、四个参数分别表示起始柱、中间柱与目的柱。如果再用过程

move(X,n,Y)

表示将 X 塔座上的 n 号圆盘移到 Y 塔座上，则可得求解 n 阶 Hanoi 塔的算法如下。

算法 1.3 求解 n 阶 Hanoi 塔问题。

```
Hanoi(n,X,Y,Z)
    IF n=1 THEN move(X,1,Z)
    ELSE
      { Hanoi(n-1,X,Z,Y)
        move(X,n,Z)
        Hanoi(n-1,Y,X,Z)
      }
    RETURN
```

这是一个典型的递归算法——自己调用自己。如果一个算法 P 显式地调用自己则称为直接递归(如算法 1.3 是一个直接递归算法)；如果算法 P 调用另一个算法 Q，而算法 Q 又调用算法 P，则称算法 P 为间接递归。

有些实际问题，既可以归纳为递推算法，又可以归纳为递归算法。但递推与递归的实现方法是大不一样的。递推是从初始条件出发，逐次推出所需求的结果，而递归则是从算法本身到达递归边界。通常，递归算法要比递推算法清晰易读，其结构比较简练。特别是在许多比较复杂的问题中，很难找到从初始条件推出所需结果的全过程，此时，设计递归算法要比递推算法容易得多。但递归算法也有一个致命的缺点，其执行的效率比较低。因此，递归最适用于写算法。

(五) 减半递推

解决实际问题的复杂程度往往与问题的规模有密切的关系。例如，两个 n 阶矩阵相乘，通常需要作 n^3 次乘法，两个二阶矩阵相乘需要作 8 次乘法。设两个二阶矩阵为

$$A=\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix},\quad B=\begin{bmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\end{bmatrix}$$

其乘积矩阵 C 为

$$C=\begin{bmatrix}c_{11} & c_{12}\\ c_{21} & c_{22}\end{bmatrix}=\begin{bmatrix}a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22}\\ a_{21}b_{11}+a_{22}b_{21} & a_{21}b_{12}+a_{22}b_{22}\end{bmatrix}$$

能不能减少乘法次数呢？如果直接考虑一般的 n 阶矩阵相乘问题，这是很困难的。但对于低阶的矩阵相乘，如二阶矩阵相乘，减少乘法次数是有可能的。若令

$$\begin{cases}x_1=(a_{11}+a_{22})(b_{11}+b_{22})\\ x_2=(a_{21}+a_{22})b_{11}\\ x_3=a_{11}(b_{12}-b_{22})\\ x_4=a_{22}(b_{21}-b_{11})\\ x_5=(a_{11}+a_{12})b_{22}\\ x_6=(a_{21}-a_{11})(b_{11}+b_{12})\\ x_7=(a_{12}-a_{22})(b_{21}+b_{22})\end{cases}\tag{1.7}$$

可以验证，乘积矩阵 C 的各元素可用以上 7 个量的线性组合表示，即

$$\begin{cases}c_{11}=x_1+x_4-x_5+x_7\\ c_{12}=x_3+x_5\\ c_{21}=x_2+x_4\\ c_{22}=x_1+x_3-x_2+x_6\end{cases}\tag{1.8}$$