第一章 计算机算法

电子计算机自 1946 年问世以来，以迅猛的速度发展，在短短的四十多年中，已经历了四个发展阶段，即所谓的“四代”——电子管计算机、晶体管计算机、集成电路计算机、大规模集成电路计算机，现在正在研制着第五代计算机。

计算机的出现是科学技术发展史上的一场伟大的革命，对人类社会产生了深远的影响。现在几乎每个领域都在大力开展计算机的应用。

计算机并不神秘，学会使用计算机也非难事。计算机是按照人们的意旨进行工作的。为了进行运算，人们必须事先准备好数据和程序。所谓程序，是指用计算机语言表示的操作步骤。例如，一个 BASIC 程序就是用 BASIC 语言写出的让计算机如何进行操作的一系列指令。

为了编好程序，就必须事先整理出解题的思路，正确地拟定出每一个操作步骤。“算法”就是研究这个问题的。本章将介绍算法、计算机和程序的有关知识。

1.1 算法与计算机

1.1.1 算法的特征

初学者往往认为计算机解题是一个不可思议的过程。其实，计算机解题的过程与操作机器、执行任务、演奏音乐、打算盘、打太极拳、炒菜、做饭等极为类似。

广义地说，做任何事情都是一个过程，这个过程可以看作为在规定条件下能够进行的基本操作所组成的序列。例如，开汽车的过程可以看作是如下一个可进行的基本操作序列：

1. 掏出钥匙；2. 打开车门；3. 坐在座位上；4. 打开电门；5. 打火；6. 加油；7. 踩离合器；……。或者说，我们在做任何事之前，都要在规定的解题环境之内，用所允许的基本操作去构造解题的步骤，只不过在做那些我们已经熟悉(习惯)的工作时，无须再有意识地考虑它罢了。例如，歌手唱歌、乐队演奏，都要为他们事先设计乐谱，乐谱就是由音符组成的序列，而音符就是音乐中所规定的基本操作的符号。同样，打太极拳的图谱、做菜的菜谱、珠算的口诀、工作计划、生产流程、治病处方、……，都代表了在不同的解题环境中，为实现某一目的所应完成的基本操作序列。只不过在不同的解题环境下，对基本操作有不同的定义。这种解题操作序列称为“算法”。这里“解题”二字是泛指解决某一问题，而不只是指“计算”。计算机科学家 D. E. Knuth 说：“一个算法是一个有穷规则的集合，其中的规则规定了一个解决某一特定类型问题的解答。”算法的概念源于数学，当然包括了在数学中的应用。

例 1.1.1 有三个数 a，b，c，找出其中最大的数。

我们先考虑处理这类问题的思路，先将 a 和 b 两数相比，将大者放在变量 max 中，然后再让 c 与 max 相比，如果 c>max，将 c 的值送到 max 中，最后 max 就是三个数中最大的数。

将它写成以下形式：

S1：将 a 与 b 比，如是 a＞b，则使 a 的值作为 max 的值，否则，使 b 的值作为 max 的值。

可以写成：如 a＞b，则 a⇒max，否则，b⇒max。

S2：将 c 与 max 比，如 c＞max，则 c⇒max。

最后 max 的值就是三个数中最大的数。

上面用 S1，S2 表示步骤的次序，在写算法时常用这种形式的标记。S 是 step(步)的首字母。S1 代表“第一步”，S2 代表“第二步”，……。

例 1.1.2 有两个变量 a 和 b，要求将它们的值互换。例如 a＝3，b＝4，互换后，a＝4，b＝3。

为了进行两个变量的值互换，须引入一个临时变量。正如两个瓶子内的液体互换需要用第三个瓶子作为过渡一样，见图 1.1。其算法可表示如下：

S1：a⇒c(将 a 的值送给变量 c)；

S2：b⇒a(将 b 的值送给变量 a)；

S3：c⇒b(将 c 的值送给变量 b)。

通过以上三个步骤实现了两个变量值的交换。这个方法在以后程序设计中常会用到。

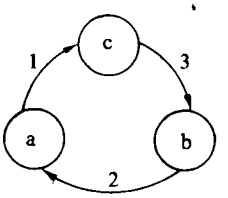


图 1.1

例 1.1.3 求 1＋2＋3＋4＋5，即求 $\sum_{n=1}^{5} n$。

先用最原始的方法进行：

S1：先进行 1＋2 的运算，相加的和(结果)放在变量 N 中(N 的值为 3)。

S2：将 N 再加 3，和仍放在 N 中(此时 N 的值为 6)。

S3：将 N 的值再加 4，和仍放在 N 中(此时 N 的值为 10)。

S4：将 N 的值再加 5，和仍放在 N 中(此时 N 的值为 15)。

可将算法写成如下形式：

S1：令 N＝1

S2：令 I＝2

S3：N 与 I 相加，结果放在 N 中

S4：使 I 的值加 1，即 I＋1⇒I

S5：如果 I≤5，返回 S3，否则算法终止

最后 N 的值就是 $\sum_{n=1}^{5} n$ 的结果。

在执行这个算法时，S3 和 S4 两个步骤要重复执行多次。直到 I＞5 为止(请读者考虑 S3 和 S4 重复执行多少次?)。可以看到，这个算法是采用循环的方法来处理的。请考虑：在 S5 步骤中的循环继续的条件为什么是“I≤5”，如果改写为“I＜5”会出现什么情况？

显然，这个算法是比较好的。如果需要求的是 $\sum_{n=1}^{100} n$，即 1＋2＋…＋100，上述算法基本上可不必改，只需将 S5 步骤中的“若 I≤5”改为“若 I≤100”即可。读者可自己写出此完整的算法，并分析它的执行过程。

例 1.1.4 有 10 个数，找出其中最大的数。

本题的思路：如同打擂台一样，台上先站着一个人，第二个人上台与之比较，胜者留在台上，然后第三个人上台再与台上的人(胜者)比较，胜者留在台上，比完九轮之后，在台上的人

便是最后胜者。

今先将输入的第一个数放在 MAX 中，将输入的第二个数与之相比，如第二个数大于 MAX，则它取代 MAX 的原值。然后第三个数再与 MAX 比，大者放在 MAX 中……，一直到比完 9 次为止。可以写成以下的算法：

S1：输入一个数，放在 MAX 中。

S2：1⇒I(I 用来累计比较次数)。

S3：输入一个数，放在 X 中。

S4：若 X>MAX，则 X⇒MAX。

S5：I+1⇒I(I 的值加 1)。

S6：若 I≤9，返回 S3 继续执行。否则，停止。此时 MAX 的值就是 10 个数中最大的数。

例 1.1.5 求两个正整数 m 和 n(m>0,n>0,m>n)的最大公约数。

求两个正整数的最大公约数的算法是介绍算法时常常引用的一个典型例子。为了使读者容易理解，我们首先看看手工求 M=60 和 n=33 的最大公约数的过程：

S1：以 n(33)除 m(60)，得余数 r(27)。

S2：判断 r 是否等于零：若 r=0，则 n 为解，若 r≠0，则进行 S3 的操作(现 r=27，故进入操作 S3)。

S3：以 n 作为新的 m(33)，以 r 作为新的 n(27)求新的 m/n 的余数 r(6)。

S4：判断 r 是否为零：若 r=0，则前一个 n 即为解；否则要继续 S5 的操作。

S5：以 n 作为新的 m(即使 27⇒m)，以 r 作为新的 n(即使 6⇒n)，求新的余数 r(3)。

S6：判断上一个 r 是否等于零：若 r=0，则前一个 n 即为解；否则执行下面的操作。

S7：再以 n 作为新的 m(6⇒m)，r 作为新的 n(3⇒n)，求新的 r(r=0)。

S8：判断上一个 r 是否等于零。这里已有 r=0，所以算法结束，n=3 即为 60 与 33 的最大公约数。

这种算法称为辗转相除。算法中使用的基本操作：整除求余和判断。它们都是进行算术运算时所允许的操作，只不过判断这一操作是在人脑中完成的，并不需要显式地写出来罢了。

从上面的算法中可以看出，S8，S6，S4 与 S2 所完成的操作是相同的，S7，S5 与 S3 所完成的操作也是相同的。或者说，S8，S6，S4 是 S2 的重复，S7，S5 是 S3 的重复。经过加工整理，可以得到下面的一个简单明了的算法 2：

S1：置数。将两个数中的大数放到 m 中，小数放到 n 中。

S2：求余。求 m/n 的余数 r。

S3：判断。若 r=0，则 n 就是所求最大公约数；若 r≠0，执行下一步。

S4：置换。使 n 值作为新的 m 值(n⇒m)，使 r 值作为新的 n 值(r⇒n)。

这个算法的 S4，使用了循环，控制 S2，S3，S4 重复执行几次，使算法变得简单明了。稍作变换，我们还可以将它变换为下面的算法 3：

S1：置数。将两个数中大数放到 m 中，小数放到 n 中。

S2：重复执行下面的序列，直到求得 r=0 为止。

S2.1：求余。求 m/n 的余数 r。

S2.2：置换。n⇒m，r⇒n。

S3：输出 m。

上面的算法对任何 m 与 n 都是适用的。要理解这个算法或要写出这个算法需要经过一定的抽象，即从具体问题出发，经过提炼、归纳，找出解决本类问题的普遍规律。譬如在智力竞赛时，对于“1＋2＋…＋100”这样一个题目，有人按①作 1＋2＝3，②作 3＋3＝6，③作 6＋4＝10，……的方法去作；而有人找到了另一种高效率的算法：100＋(1＋99)＋(2＋98)＋…＋(49＋51)＋50，共得 50 个 100，1 个 50，即和为 5050。一旦得到这样一种方法，当遇到题目为 1＋2＋…＋1000 时，他决不会重新冥思苦想地另寻算法，而是运用前面的方法，很快算出 1000＋(1＋999)＋(2＋998)＋…＋(499＋501)＋500＝500500。

例 1.1.6 求正整数 A 和 B 的积，A 和 B 为任意位数。

先看一下人工计算是怎样进行的。假如 A＝3412，B＝513，乘法竖式为

```
  3413——A
 ×513——B
 10236
  3412   } 部分积
17060
1750356——积
```

它们是这样进行的：以 A 为被乘数，分别以 B 的个位数、十位数、百位数作为乘数，求出它们的部分积。个位数与 A 相乘，得到第一行部分积，十位数与 A 相乘得到部分积(第二行)应左移一位，或者认为将该位数字乘以 10 再和 A 相乘；同样由乘百位数得到的第三行部分积应左移二位，或者认为将该数乘以 100 再和 A 相乘。把各部分积相加即得最后的积。

现在 A 和 B 的位数都是任意的，那么怎样用一般抽象的方法表示此算法呢？

将乘数 B 表示为 $b_nb_{n-1}\cdots b_2b_1b_0$，$b_0$ 表示个位数字，b_1 表示十位的数字，b_2 表示百位的数字，其余类推。用 C 表示某一时刻的部分之和。用 i 表示乘数的位数，i＝0 表示个位(因为 $10^0=1$)，i＝1 表示十位(因为 $10^1=10$)，i＝2 表示百位(因为 $10^2=100$)，……。

算法可以表示如下：

S1：使 i＝0，C＝0(未进行乘数运算前，部分积之和为零)，并给 n 赋初值。

S2：当 i≤n 时，重复执行下面的操作：

S2.1：$C+A\times b_i\times 10^i\Rightarrow C$(乘 10^i 即左移 i 位，第一个部分积为 $0+A\times b_0\times 10^0=A\times b_0$)；

S2.2：i＋1⇒i(i 增 1)。

S3：输出 C。

请读者按上述算法进行 853×28654 的运算。

例 1.1.7 求两个整数 X(X≥0)和 Y(Y＞0)的整数商和余数(规定只能用加法和减法运算)。

除法运算实际上是用加法和减法来实现的。其办法是，从 X 中一次又一次地减去 Y，直到 X＜Y 为止，所减的次数即为商。例如 5÷2 的商为 2，余数为 1。可以将 5 先减 2，余 3，再减一次 2，余 1。每减一次，商加 1。5 能减两次 2，所以商就得 2。此时余数为 1，不能再减 2，结束。用 Q 代表商，R 代表余数，算法可以表示为

S1：使 Q＝0，R＝X(在开始减以前，商为零，所余的数就是 X 本身)。

S2：当 R≥Y 时，重复下面的操作：

S2.1：R－Y⇒R；

S2.2：Q＋1⇒Q。

S3：输出 R。

上面我们举了几个数学算法的例子。每一个算法都由加、减、乘、除、判断、置数等基本操作，按顺序、判断、分支、重复等结构组成。一般说来，任何复杂的数学问题都可以由上述这些最基本的操作按一定的结构组成的算法来求解。研究算法就是研究怎样把各种类型的问题的求解过程分解成上面的基本操作。

实际上，算法可以看作与演绎平行的另一数学体系。

由前面的讨论可知，算法是对解题过程的抽象和精确描述。一般来说，一个算法应当具备以下几个特征：

1. 有穷性：一个算法应当包括有限个操作步骤，或者说它是由一个有穷的操作系列组成，而不应该是无限的。例如，创作一首小提琴曲谱可以称为小提琴算法。如果有人说他设计一首永远演不完的小提琴曲谱，人们将无法演奏完这个曲子，因而它不能称为算法。

不仅如此，有穷性还常常理解为实际上能够容忍的合理限度。例如，执行一个计算机算法需要让计算机运行 10000 年，便是不能容忍的。

2. 确定性。算法中的每一步的含义都应该是清楚无误的，不能模棱两可，也就是说不应该存在“歧义性”（即可作两种以上不同的解释）。请读者分析下面这句话：张三对李四说他的孩子考上了大学。这句话就是“歧义性”的，可以理解为张三的孩子考上了大学，也可理解为李四的孩子考上了大学。因此在表示算法时要使用明确的文字或数字语言。

3. 一个算法应该有零个或多个输入。如例 1.1.5（求最大公约数）中，有两个输入量 m 和 n。有的算法也可以没有输入。例如一盒音乐磁带，只要启动，就会放出音乐，而没有输入。

4. 一个算法应该有一个或多个输出。算法的目的是求解，“解”就是输出。对无解的问题也应给出“无解”的信息。没有输出的算法是没有意义的。例 1.1.5 中的输出是最大公约数。

5. 有效性。即我们在前面已经提到的一个算法必须遵循特定条件下的解题规则，组成它的每一个操作都应该是在特定的解题规则中允许使用的、可以执行的，并且最后能得出确定的结果。如在例 1.1.5 中给定的 n 为零时，执行第一步便会遇到一个在算术运算的条件下是无意义的运算（分母为 0）。只要算法中有一个操作是不可执行的，整个算法就不具有有效性。

基于算法的特征，还可以给出算法的另一种定义：算法是一个过程，这个过程由一套清楚的规则组成，这些规则指定了一个操作顺序，以便用有限的步骤提供特定类型问题的解答。这里需要指出的是，并非所有的过程都能称为“算法”。过程可以不受“有穷性”的约束，即它的操作可以是无穷尽的。例如求所有自然数之和，它要执行无穷尽的加法，那么这个工作便是一个“过程”，而不能称为算法。

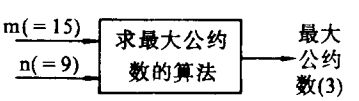


图 1.2

对使用算法的人（而不是设计算法的人）来说，可以把一个算法看作一个“黑箱”，给它输入某些确定的量，它就会输出确定的输出量。如求 m 和 n 最大公约数的算法的“外部特性”可以用图 1.2 表示。也就是说，从算法的外部（而不是研究算法的内部）看，它的作用是：给它输入 m 和 n，

它就能输出一个最大公约数。从此例也可以理解算法的输入和输出的概念。

1.1.2 计算机——实现算法的有力工具

前面我们讨论了算法的基本概念，但是算法只是指出操作的内容和步骤，或者说只给出了实现目标的过程描述，要把它付诸实践得到预期结果，还需要解决工具问题，即用什么工具去实现算法。正如用珠算算题，口诀是算法、算盘是工具一样。只有口诀没有算盘，就难以有效地算题。再如，作曲家写出一首名曲，用五线谱形式发表，这就是算法。但是五线谱不会发声，需要某种乐器，按照乐谱演奏出来，这就叫做"算法的实现"。最古老的算法工具是人工方式。用人工方式处理一些简单的算法，还能在可容忍的时间内完成，而对于复杂的算法，则往往人工难以在可容忍的时间范围内完成。例如，国外有人用手工计算 π 值，计算到小数点后的 707 位，花了十五年时间。如果要计算到小数点后的 5000 位，恐怕一辈子也完不成。

人类从远古时代便开始制造工具，以放大或延伸自身的能力。计算工具是一种智力工具。从棍石记事、屈指计算开始，人类先后制造出了算筹、算盘、计算尺、手摇计算机、电动计算机等计算工具，用以"放大"自身的智能，提高智力劳动的效率。1946 年，世界上第一台电子计算机 ENIAC 诞生了。这是当时已经相当成熟与丰富的计算理论与电子技术相结合的硕果，也是当时正在迅猛发展的科学技术的迫切需要。ENIAC 是在美国陆军部主持下，由宾夕法尼亚大学电子系工程师 J. P. Eckert 和物理学家 J. W. Mauchly 等人研制成功的。它的运算速度比当时已有的其它类型的计算机提高了 1000 倍，每秒钟可进行 5000 次加法、300 次乘法。40 年代末以来，电子计算机一直在突飞猛进地发展，已经经历了电子管、晶体管、集成电路、大规模集成电路等四代。运算速度已可达到每秒亿次以上。这样高的运算速度使其之前的任何计算工具及人的智力所望尘莫及，也使得我们能在允许的时间内完成过去无法实现的许多算法，如前述要把 π 的值计算到小数点后面 5000 位等问题，都可以在短时间内实现。

然而，电子计算机的优势还并不单单在于其运算速度高上。它区别于其前那些计算工具的另一惊人成就是它能自动地执行操作。这一成果主要归功于一位美籍匈牙利数学家冯诺曼(Von Neumann)。

1944 年夏天，正在普林斯顿工作的冯诺曼偶然地得到了正在研制 ENIAC 的消息，并引起了他的极大兴趣。他很快发现 ENIAC 的致命缺点，提出了以二进制和程序存储控制为基础的计算机体系结构思想。

二进制原理以二元逻辑为基础。它要求所有信息在机器内部都用 0,1 两个码的组合表示。表 1.1 是几个普通十进制数的 0,1 码表示。它遵循逢二进一的规律，也称二进制数。二进制原理确立了电子计算机经济而实用的物理结构。

程序存储控制理论使算法的自动执行成为可能。程序是以机器能理解的信息描述的算法。以前，程序被保存在机器外部，机器要由人按程序的规定步骤一步一步地操作，如用算盘进行计算时，每一步运算都要由人按口诀内容拨动珠子进行。这就把人与机器捆在了一起，不仅把人束缚在繁冗的运算过程之中，而且难以发挥出机器运算速度高的这一突出优势。程序存储控制原理就是要让机器"记住"程序，并按程序规定的步骤控制自己的工作。这样便可以不需要人再去干预运算过程，解放了人，同时使机器的运算速度高的优势得到充分发挥。

现代计算机基本上都是按冯诺曼原理制造的，所以也称为冯诺曼型计算机。按照程序存

表 1.1

十进制数	二进制数	十进制数	二进制数
0	0	5	101
1	1	6	110
2	10	7	111
3	11	8	1000
4	100	9	1001

储控制原理，现代计算机应由输入设备、输出设备、运算器、控制器、存储器等五大部件组成，如图 1.3 所示。当人用输入设备把程序和数据送到机器后，机器就把它们保存到存储器中。执行时，控制器从主存储器中依次取出程序指令，并逐条进行分析，根据指令的要求控制运算器对指定的数据进行相应的运算。控制器和运算器是计算机的心脏，由它进行控制与操作，它称为中央处理器(Centre Processing Unit)，简称 CPU。

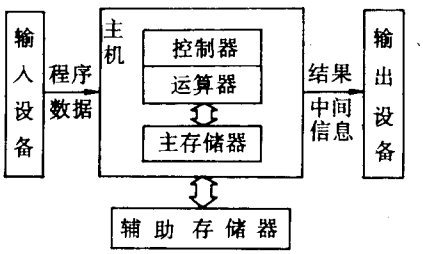


图 1.3

输出设备能将运算的中间信息(如出错、中间结果等)和最终结果输出到某个输出设备上。常用的输入设备是键盘，输出设备是显示器和打印机。

在计算机的全部设备中，存储器是一个很重要的设备。就像人没有记忆功能，就什么也做不成一样，计算机没有存储器，就无法存储程序和数据，就不能进行程序存储控制，自动工作也就无法谈起。

二进制电路是存储器的“细胞”。每个“细胞”可以存储一个二进制信息码，要么是“1”，要么是“0”，称为一“位”(b，即 bit 的缩写。一个存储器可以存储亿万个二进制信息。为了管理方便，现代计算机把几个二进制信息组织在一起，称为一个“字节”(B，即 byte 的缩写)。一般一个字节由八位组成。存储器的容量一般以字节为单位，如长城 GW0520DH 的存储器可供用户使用的容量为 640KB(1K＝2^{10}＝1024)，而长城 GW286B 为 1MB(1M＝2^{20}，约为 1 兆)，IBM PC 为 64KB，IBM PS/2-30 为 640KB，IBM PS/2-80 为 2～16MB。计算机中的一个逻辑信息，即一条指令或一个数据，称为一个计算机字(word)。一个字所包含的二进制位的个数称为字长。通常字长是字节的整数倍。如 True BASIC 规定每个数值数据占 8 个字节。

多数计算机是以字节为单位进行信息存储的。存储一个数据的存储空间，称为一个存储单元。一个存储单元可以包括一个字节或几个字节。为了能向指定的存储单元存入或取出指令或数据，需要对每一个存储单元编号，如同旅馆中的房间号一样。存储单元的编号，称为该单元的“地址”。按地址检索信息，是当代计算机存储器的一大特征。即向存储器中存入一个信息时，应指出把该数据存入哪个单元(指出地址)，而从存储器中取一个数据时，是指取出哪个单元内的数据，而不是指取出哪个数值。如图 1.4 所示，假设在 2001 单元内已放入了数据“2”，在 2002 单元中已放入了数据“3”，如果想进行 2＋3 的运算，应说“将 2001 单元中的内容与 2002 单元中的内容相加，送到 2003 单元中”，记作

(2001)＋(2002)⇒2003

(2001)表示 2001 单元中的内容，而 2003 指单元地址而不是内容。请读者不要混淆单元

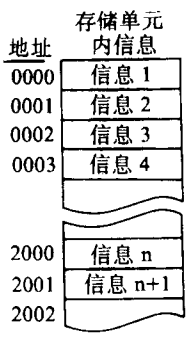


图 1.4

地址与单元内容。正如不要混淆旅馆中的某房间与该房间中的旅客一样。上式中不加括弧的数字指的是单元地址，加括弧的指的是单元内容。一个存储单元中存入一个信息后，只要不再向它里面存入别的信息，原有的信息便会一直保存，可供多次“取用”，其值不变。就像在磁带中录入一首歌曲后，只要不用它录制别的歌曲，原来的歌曲便会一直保存，可供反复欣赏一样。而一旦重新存入了别的信息，原来的信息便不复存在了。如进行这样的操作：

$$(2001)+(2002)\Rightarrow 2001$$

操作结束后，2001 单元内的数据便不再是 2，而成为 5 了。存储器的这一特点可通俗地称为“取之不尽，新来旧去”。这种存取与向一个仓库中存取货物有本质区别（仓库中每经一次存取之后，物品的数量是变化的，而从存储单元中“取数”后，单元中内容不变），因此，把信息的存取称为“写入”或“读出”更为确切、形象。

容量、存取速度和成本是评价计算机存储器功能的三项主要指标。但这三项指标往往是互相矛盾的。例如，容量大，存取时为寻找某一单元所花费的时间就多；成本高，就难以做得容量很大。为此，现代计算机采用“分级存储”技术来解决这一矛盾。通常采用两级存储方式，即把存储器分为主存储器（即内存储器）和辅助存储器（即外存储器）两级。对内存的存取速度高，通常用以存储当前需处理的信息。对外存的存取速度低，但价格便宜，可以做得容量很大。计算机工作时，内存与外存之间可以成批交换数据。目前内存基本上采用半导体存储器，外存多用磁带、磁盘等磁表面存储器。半导体存储器是一种“易失型”存储器，断电后原来存储的信息会全部丢失。而磁介质存储器的信息则不会因断电而消失。可以长期保存信息。顺便说明一下，计算机存储器中有成千上万个存储单元，要在程序设计时给指令和数据一一分配存储单元是一件十分烦琐的工作。这项工作现在已经可以交给一种软件——操作系统去完成了。这样，程序员就可以有可能用一个名字（即变量）来代替地址编写程序了。这样 2+3 便可以写成

$2\Rightarrow A$

$3\Rightarrow B$

$A+B\Rightarrow C$

其中“⇒”代表往存储单元中送数据。

1.1.3 计算机科学是研究算法的科学

输入设备、输出设备、运算器、控制器、存储器分别承担了计算机的输入、输出、运算、自我控制和记忆等功能。然而这些物理部件（也称计算机硬件）只不过是机器的躯体。计算机的全部工作实际上是在程序的控制下进行的。或者说这些物理部件是在程序的支配下工作的。所以人们常把程序称为计算机的灵魂。然而，这个灵魂是人给的，人就是计算机的“上帝”。

计算机程序就是用计算机能够理解的信息（即计算机语言）描述的算法。为了让计算机正确、可靠而高效地完成解题任务，需要设计一个好的算法，然后根据此算法编写程序。因

此，从某种意义上说，算法的研究就成为计算机科学的核心。当然，这并不是说，只有有了计算机才有了算法科学。其实算法是一门很古老的科学。

在古代，公理系统还没有发展起来的时候，人们只会利用算术方法解题。不论什么问题都要用算术方法求解，这就迫使人们不得不去精心地构造各种算法，使得以算术为代表的算法科学曾一度得到了高度发展。但是，随着公理系统的发展，人们把注意力和兴趣渐渐地转向以公理系统为基础的、内容丰富的、结构复杂的演算，而把算法——这种在人类史上创造过灿烂的文明之光的数学思想看作是“笨”办法逐步打入冷宫。

随着自动机器，尤其是电子计算机的出现，人们又开始注意算法的研究，使算法获得了新生。

由于计算机运算的速度极高，而且从根本上来说计算机只能执行一些最简单的操作(计算机所能完成的所有复杂的运算都是由一系列简单的操作组成的)，因此人们必须利用计算机高速操作这一特点去完成一系列简单而“笨”的操作。这就不得不重新认识算法在现代条件下的价值，深入地进行算法的研究。我们面临的任务：用计算机能够执行的基本操作去组成解题步骤——算法。或者说，是要把一个算法分解为计算机可以执行的基本操作。

计算机虽然是用现代科学技术武装起来的高技术产品，它的基本部件却是只能表示 0 或 1 的电子开关。计算机能执行以下的基本操作：

1. 数据传送：由输入设备把数据送到内存；由内存把数据传送到输出设备；由内存一个单元把数据传送到另外一个单元。
2. 逻辑运算：“与”、“或”、“非”。
3. 算术运算：加、减、乘、除、乘方等。
4. 比较、判断和转移。

要靠这些简单的操作的组合去解决复杂问题，迫使人们不得不花很大的力气去研究如何用计算机解题的计算机算法。

通常把计算机处理的问题分为两大类：一类是“数值运算”，如解一个联立方程；求一个函数的定积分等。另一类是：“非数值运算”，例如，有一批人名要按它们的汉语拼音字母排序；用计算机检索图书馆中的书刊等。过去有的人以为计算机的作用只是进行数值计算，这是一个误解。当今，计算机已深入到人类生产和生活的各个领域，在非数值领域的应用远远超过了数值运算领域。50 年代初期，计算机在数值计算方面的广泛应用，使得讨论计算方法一类算法的计算数学迅速发展并成熟。50 年代中后期，计算机开始广泛应用于非数值运算领域。非数值运算的特点是其处理对象大多数是非连续性的数据(例如图书馆中书目数据是不连续的)，这种数据称为“离散性”数据。非数值运算领域的应用促进了以研究离散对象为目标的组合算法的研究。

1.2 算法的表示

1.2.1 概述

前面述及，算法是解题方法的精确描述。描述算法的工具对算法的质量有很大的影响。一般说来，可以使用三种类型的描述工具来表示算法。

1. 自然语言：人们日常使用的语言，如汉语、英语、日语等。上节中的例子基本上都是用汉语描述的算法。使用自然语言描述算法，人们比较习惯，容易接受，但它有以下缺点：

(1) 自然语言容易有“歧义性”，不太严格。例如前面说到的“张三对李四说他的孩子考上了大学”这句话就不严格，具有歧义性。

(2) 用自然语言描述比较冗长。例如：“置 m 的值为 n，置 n 的值为 r”，不如表示成“n⇒m，r⇒n”简洁。

(3) 自然语言的表示形式是顺序的，即一句一句地顺序地叙述下来，或者说各步骤是串行的。如果算法中有分支或转移时，用文字表示就显得不那么直观。

因此，自然语言不是一种好的算法描述工具，只有在不致引起混乱的情况下，才可以用简洁的文字来描述某一步骤的操作。

2. 专用工具：为了更好地、准确地描述算法，人们创造了许多专用的算法工具。这些算法工具有使用图形的(如流程图，结构化流程图，PAD 图，IPO 图，Warnier 图等)，也有使用表格的(如判定表等)，还有使用代码符号的(如伪代码)。

3. 计算机程序设计语言：计算机程序设计语言是可以让计算机接受、理解和执行的算法描述工具。任何计算机算法最终都要用程序设计语言描述出来，才能在计算机上实现。用计算机程序设计语言描述的算法的形式也是串行的。串行形式难以直观地反映算法的逻辑结构，例如当有转移、分支、循环时，直接看程序就不那么直观，所以在算法设计阶段人们多用专用描述工具。下面介绍两种常用的算法的描述工具——流程图和 N-S 结构化流程图。

1.2.2 流程图

流程图是一种算法的图形描述工具。它用一些几何图框表示各种类型的操作，在框内写上简明的文字或符号表示具体的操作，用箭头的流程表示操作的先后顺序。图 1.5 为 ANSI(美国国家标准化协会)规定的一些常用的流程图符号。

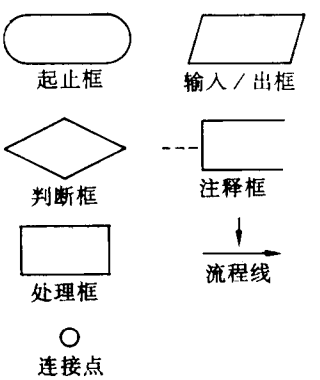


图 1.5

下面是用流程图表示算法的例子。

例 1.2.1 用流程图表示例 1.1.1 的算法(找出三个数中最大的数)，见图 1.6。

与例 1.1.1 所列的算法相比，增加了“输入 a，b，c”和输出 max 值的两个框，以适应计算机算法的特点。

例 1.2.2 用流程图表示例 1.1.2 的算法(两数互换)，见图 1.7

例 1.2.3 用流程图表示例 1.1.3 的算法 $\left(\text{求}\sum_{n=1}^{5}n\right)$，见图 1.8。

例 1.2.4 用流程图表示例 1.1.4 的算法(找 10 个数中的最大数)，见图 1.9。

例 1.2.5 用流程图表示例 1.1.5 的算法(求最大公约数)，见图 1.10。

在图 1.10 中用了一个“注释框”，用来声明“m，n 应为正整数”。

例 1.2.6 用流程图表示例 1.1.6 的算法(求 A 和 B 的乘积)，见图 1.11。

输入 A，B 的值和 B 的位数，例如，B=513，则 m=3。n=m−1=2。B 可以表示为

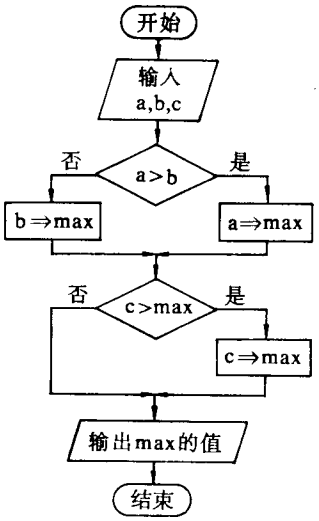


图 1.6

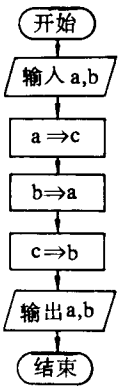


图 1.7

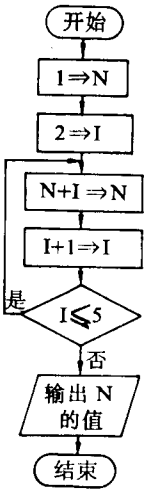


图 1.8

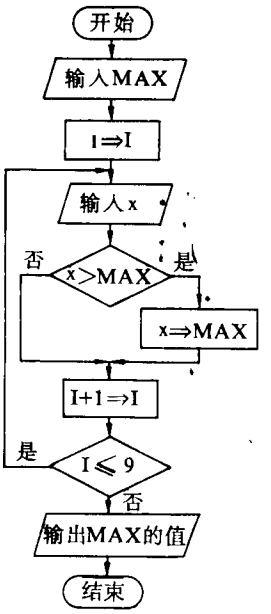


图 1.9

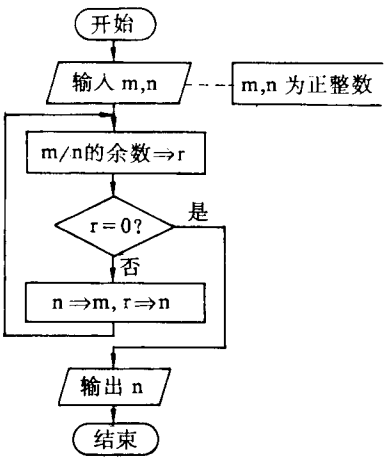


图 1.10

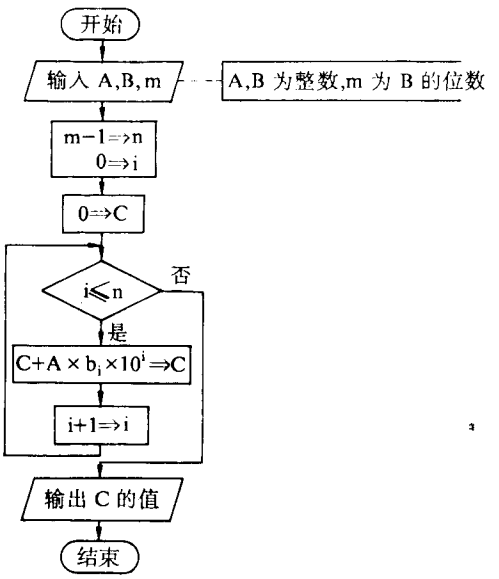


图 1.11

$$B=b_2b_1b_0=5\times10^2+1\times10^1+3\times10^0$$

图 1.11 中的循环部分共执行 3 次。

1.2.3 三种基本结构

用流程图表示算法方便直观，画图方法比较自由灵活，但是它存在着一个致命的弊病：

由于流程图要用带箭头的流程线指明执行的先后顺序，对流程线的使用也无任何限制，即允许从某一处用流程线将流程转向流程图中的任何一个框的入口处，看起来方便，但随心所欲不加任何限制地在流程图中用流程线使流程无规则地转来转去，由于问题的规模和复杂程度的增加，将使流程图变得象乱麻一样，使人难以理解。或者说，这种算法结构没有规律、清晰度差，使人难以阅读。图 1.12 是这种算法结构的示意图，其中每一根横线代表一个框的功能。由于这种算法结构象“面条”一样（象一碗面条一样互相缠绕在一起，难以找到头和尾），故被称之为 BS(Bowl of Spaghetti)型，也有的称其为耗子窝(rat's nest)型。

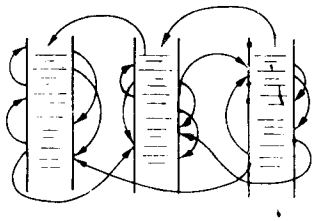


图 1.12

显然，这种 BS 型结构不利于阅读和交流，也不便于修改，从而使算法的可靠性和可维护性难以保证。为了提高算法的质量，必须限制箭头的滥用，使算法结构规范化。

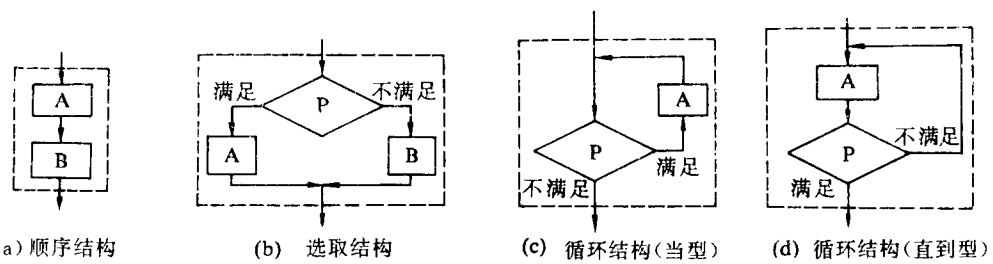
1966 年 Bohra 和 Jacopini 提出了结构化算法的三种基本结构单元：

(1) 顺序结构；

(2) 选取结构(或称选择结构)；

(3) 循环结构(或称重复结构)。

这三种基本结构的流程图，如图 1.13 所示。



(a) 顺序结构　(b) 选取结构　(c) 循环结构(当型)　(d) 循环结构(直到型)

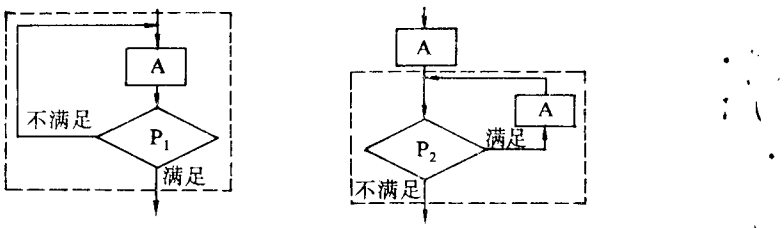
图 1.13

图 1.13(a)表示顺序结构，要求算法应按书写的顺序执行，即先执行 A 操作，再执行 B 操作。

图 1.13(b)表示二路分支的选取结构，要求执行过程中根据条件 P 是否满足选择执行 A 操作或 B 操作。

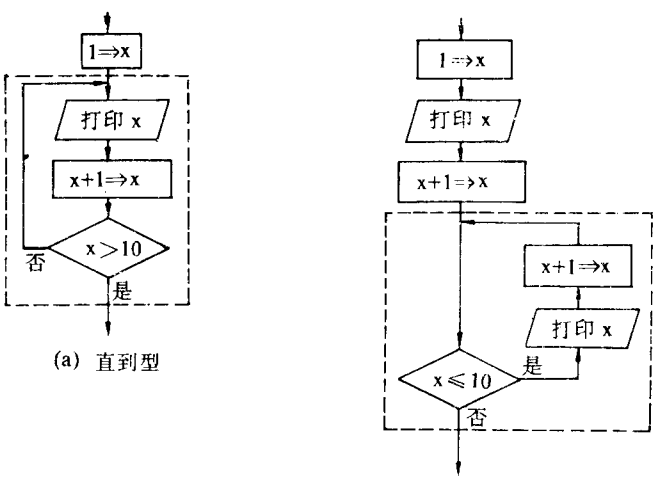
图 1.13(c)与(d)是循环结构的两种形式。图 1.13(c)称为“当型”(或 WHILE 型)循环，它要示当条件 P 满足时执行 A 操作。显然，当一开始条件 P 就不满足时，A 操作不会被执行。图 1.13(d)称为“直到型”(或 UNTIL 型)循环，它要求执行操作 A，直到条件 P 满足为止。显然，不管条件 P 满足还是不满足，A 操作至少要执行一次。所以当要把一个如图 1.14(a)所示的直到型循环结构转换成当型循环结构时，应先在循环结构之外复制一个 A 框，然后将条件 P_1 改为 P_2，P_2 是 P_1 的“反条件”，即 $P_2=\overline{P}_1$，如图 1.14(b)所示。例如，图 1.15(a)的直到型循环结构可转换为图 1.15(b)所示的当型循环结构。请注意，图 1.15(a)中的判断

条件为“x＞10”，而图 1.15(b)中的判断条件为“x≤10”，二者互为反条件。同时应注意菱形判断框两侧的“是”和“否”的位置恰好相反，直到型循环(图 1.15(a))是条件满足时不再重复执行当中的两个框(循环体部分)，而当型循环(图 1.15(b))则是在条件满足时执行该两个框。



(a) 直到型　　(b) 当型

图 1.14



(a) 直到型

(b) 当型

图 1.15

从图 1.13 可以看出，这三种基本结构具有如下特点：

(1) 只有一个入口。

(2) 只有一个出口。

(3) 每一个框内的功能都有机会被执行。即对每一个框来说，应当有一条从入口到出口的路径通过它。图 1.16 就不具有这样的性质。

(4) 不包含死循环(即无休止的循环)。图 1.17 就不具有这样的性质。

已经证明，任何复杂的问题，都可以用以上三种基本结构顺序地构成。当然，除此三种结构外，算法还可以有其它结构。但是将算法结构限制为只使用上述三种基本结构后，可以使算法的各基本单元(即基本结构)之间形成顺序执行关系。每一个基本结构具有单入口单出口的性质，使不同基本结构之间的关系简单，互相依赖性较少，它们是互相独立的“元部件”，

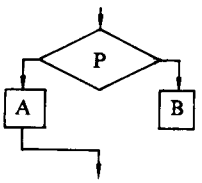


图 1.16

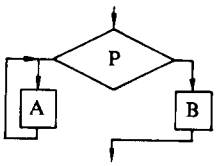


图 1.17

人们能够像搭积木一样将它们搭成各种不同的大结构。整个算法看起来就像项链上的一串珍珠一样由各基本结构顺序组成，结构清晰，从而可以使算法的质量显著提高。符合这一原则的流程图称为结构化的流程图。如图 1.18 就是一个结构化的流程图。因为它可以看作由 J,I,H 三个基本结构串接而成。而 I 是由 K,L 与 P_1 组合而成的选取型结构；K 是由 M,N 与 P_2 组成的选取型结构；M 与 N 都是基本结构。L 也是由基本结构组合而成的。

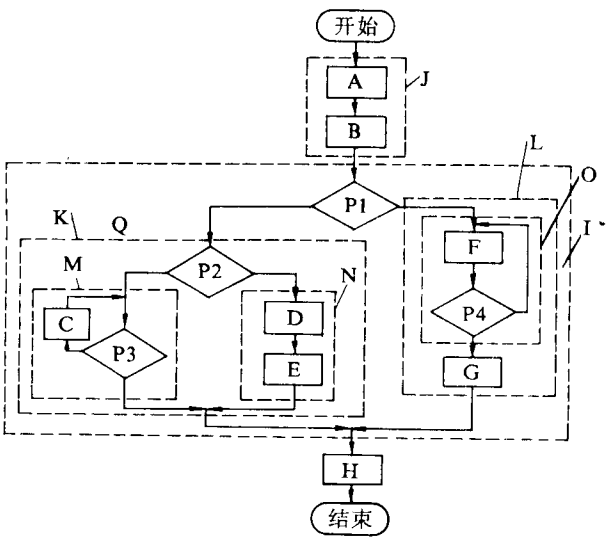


图 1.18

1.2.4 N-S 结构流程图

传统的流程图允许箭头随意转移，用这种流程图不能保证流程图是结构化的，而且这种流程图占篇幅比较大，作图工作量也较大。

1973 年美国学者 I. Nassi 和 B. Shneiderman 提出了一种新的流程图工具。由于他们二人的名字是以 N 和 S 开头的，因此，这种流程图称为 N-S 图。N-S 图完全去掉了在算法描述中引起麻烦的带箭头的流程线，全部算法写在一个大的矩形框内。规定了一些图形元素，以表示三种基本结构，每个元素用不同的框来表示。或者说，一个算法框是由一些代表基本结构的基本框象堆积木一样构造而成的。由于 N-S 图像一个多层的盒子，也称盒图(box diagram)。图 1.19 示出了它的三种基本结构图。

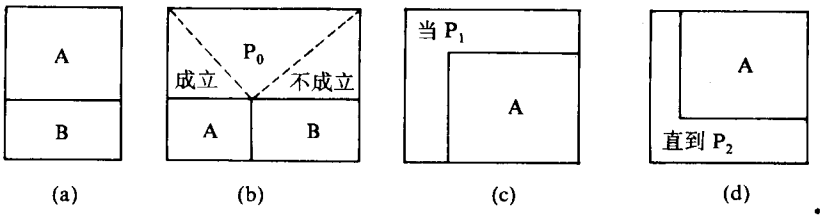


图 1.19

1. 顺序结构：如图 1.19(a)所示，其中 A 和 B 分别代表一个基本操作(例如加、减、乘、除运算，打印、赋值、……)。两个或多个矩阵框顺序组成一个顺序结构。

2. 选取结构：如图 1.19(b)所示。它表示当 P_0 条件成立时执行 A 操作，P_0 条件不成立时执行 B 操作。图 1.20 是打印出 a 的绝对值的算法描述。

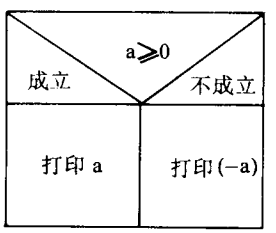


图 1.20

3. 循环结构：

(1) 当型循环：用图 1.19(c)的形式表示。当型表示当 P_1 条件满足时执行 A 操作，然后再返回判断 P_1 条件是否满足，如满足再执行 A，……，如此重复下去，直到 P_1 条件不满足为止。

(2) 直到型循环：见图 1.19(d)。它表示，先执行 A 操作，再判断 P_2 条件是否满足，如不满足则返回再执行 A 操作，如满足则不再继续执行循环。

图 1.19(c)(d)中，“当 P_1”和“直到 P_2”，也可以写成“while P_1”和“until P_2”。有时也可以省写“当”、“直到”、“while”、“until”，直接写“P”。从图案的形状即可知道它是“当型”还是“直到型”结构。前已述及，这两种结构是可以互相转换的。

当型循环与直到型循环的执行过程略有差异。由图 1.21 可以形象地看出：当型循环中，流程从上方进入本结构，先遇到 P_1(图中以①表示)，判断 P_1 条件，再往下执行 A 操作(图中以②表示)，然后再返回判断 P_1(以③表示)，……，到某次判断 P_1 不再满足，就脱离本结构，执行下一个结构(图中以④表示)。

从图 1.21(b)可以看到：直到型循环结构则先执行 A(图中①)，然后判断 P_2 条件(图中②)，如 P_2 不满足再返回执行 A(图中③)，周而复始，直到 P_2 成立，则脱离本结构(图中④)。

在一种基本结构中还可以包括另一个基本结构。如图 1.22 中，包含两个框，第二个框是

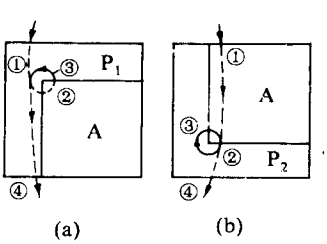


图 1.21

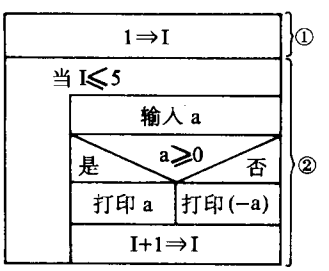


图 1.22

一个当型循环。在该当型循环中，被重复执行的部分（循环体）又包括了三个框，其中有一个选取结构。这个算法的目的是：先后输入 5 个数，打印出每一个数的绝对值。I 的作用是用来累计“次数”，处理完 5 个数之后，I＝6，不再执行循不。

可以将图 1.22 中的选取结构用一个简单的矩形代表，见图 1.23。实际上，任何一个基本结构都可以用一个简单的框代表：图 1.23 是一个“粗流程图”，图 1-22 是“细流程图”。开始设计时可以粗一些（如图 1.23），在详细设计时，再将“打印 a 的绝对值”细化为图 1.22 中的选取结构。图 1.23 中当型循环中的三个框又可组成一个顺序结构。因此，它就是图 1.19 (c)所示的当型循环形式。

用 N-S 图表示例 1.11 到例 1.16 中的算法，见图 1.24～图 1.30。

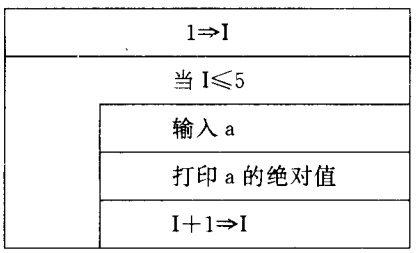


图 1.23

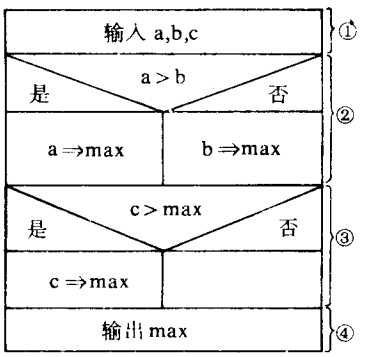


图 1.24

例 1.2.7 图 1.24 表示的是从三个数中找最大数的算法。图 1.24 的 N-S 图由 4 个框组成，第二、四两个框是选取结构。可以明显地看出，这个算法是结构化的，它是由基本结构组成的。执行此算法时，从上到下逐个执行各个基本结构。

例 1.2.8 表示交换两个变量值的算法的 N-S 图，如图 1.25 所示。

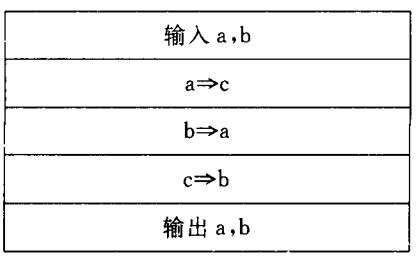


图 1.25

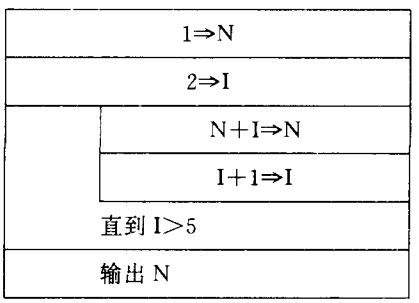


图 1.26

例 1.2.9 表示 $\sum_{n=1}^{5} n$ 算法的 N-S 图，见图 1.26。

请将图 1.26 与图 1.8 对比。图 1.26 中用直到型循环形式，结束循环的条件是 I＞5，而图 1.8 中表示继续执行循环的条件是 I≤5，当 I≤5 条件的判断结果为否（即 I＞5）时，就不再执行循环了。图 1.26 和图 1.8 表示的是同一件事，只是表示方法不同而已。也可以将图 1.8 中的菱形框内的条件改为 I＞5，同时将两个出口的“是”和“否”颠倒一下，作用是一

样的。

例 1.2.10 输入 10 个数，打印出其中最大的数。其 N-S 图见图 1.27。

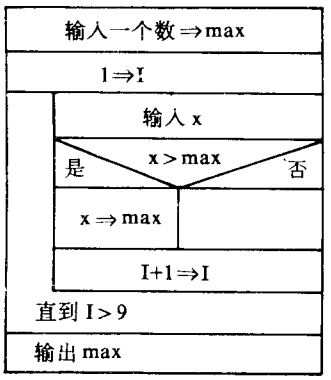


图 1.27

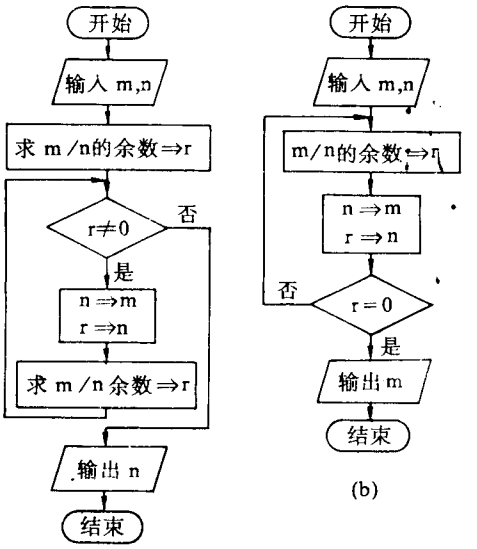


图 1.28

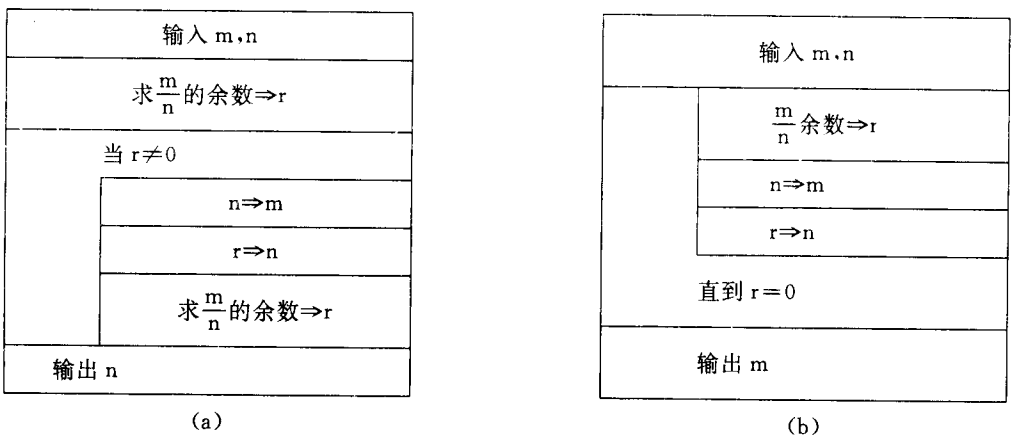


图 1.29

例 1.2.11 求 m,n 的最大公约数。

先看一下图 1.10，其中的循环不是标准的当型或直到型循环，可先将其改画成图 1.28(a)。

图 1.28(a)是一个当型循环。可以将它改画成 N-S 图形式，见图 1.29(a)，也可画成直到型循环，见图 1.28(b)和图 1.29(b)。

例 1.2.12 求两个正整数 A 和 B 的乘积。其 N-S 图见图 1.30。