的特性，就一定能想出算法。思考算法也可以是一件非常有趣的事。下面，笔者将介绍思考算法时的要点。请诸位务必以此为契机，和算法成为朋友，体味思考算法所带来的乐趣。

## 5.2　要点 1：算法中解决问题的步骤是明确且有限的

先正式地介绍一下什么是算法吧。用英汉辞典去查 algorithm 的意思，得到的解释是“算法”。诸位会觉得这个解释很含糊，不知所云吧。

再去查查 JIS（日本工业标准），上面写着算法的定义是“被明确定义的有限个规则的集合，用于根据有限的步骤解决问题。例如在既定的精度下，把求解 sin x 的计算步骤无一遗漏地记录下来的文字”。这个定义虽然看起来晦涩，但是正确地解释了什么是算法。

如果用通俗易懂的语言来说，算法就是“把解决问题的步骤无一遗漏地用文字或图表示出来”。要是把这里的“用文字或图表示”替换为“用编程语言表达”，算法就变成了程序。而且请诸位注意这样一个条件，那就是“步骤必须是明确的并且步骤数必须是有限的”。

接下来先举一个具体的例子，请诸位想一想解决“求出 12 和 42 的最大公约数”这个问题的算法。最大公约数是指两个整数的公共约数（能整除被除数的数）中最大的数。最大公约数的求解方法应该在中学的数学课上学过了。把两个数写在一排，不断地寻找能够同时整除这两个整数的除数。最后把这些除数相乘就得到了最大公约数（如图 5.1 所示）。

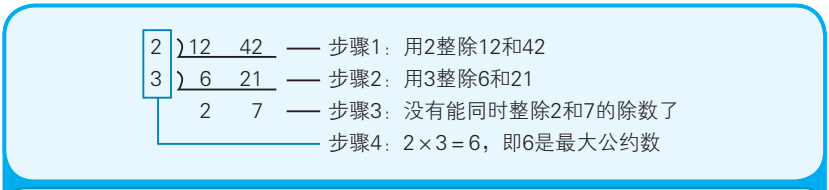



图 5.1 在中学学的求解最大公约数的方法

用这个方法求出了 6 是最大公约数，结果正确。但是这些步骤能够称为算法吗？答案是不能，因为步骤不够明确。

步骤 1 的“用 2 整除 12 和 42”和步骤 2 的“用 3 整除 6 和 21”，是怎么知道要这样做的呢？寻找能够整除的数字的方法，在这两步中并没有体现。步骤 3 的“没有能同时整除 2 和 7 的除数”，又是怎么知道的呢？而且，到此为止无需后续步骤（即步骤数是有限的）的原因也是不明确的。

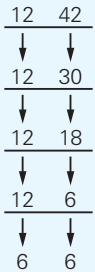
其实这些都是凭借人类的“直觉”判断的。在解决问题的步骤中，有了与直觉相关的因素，就不是算法了。既然不是算法，也就不能用程序表示了。

## 5.3 要点 2：计算机不靠直觉而是机械地解决问题

计算不能自发地思考。因此计算机所执行的由程序表示的算法必须是由机械的步骤所构成。所谓“机械的步骤”，就是不用动任何脑筋，只要按照这个步骤做就一定能完成的意思。众多的学者和前辈程序员们已经发明创造出了很多机械地解决问题的步骤，这些步骤并不依赖人类的直觉。由此所构成的算法被称为“典型算法”。

辗转相除法（又称欧几里得算法）就是一个机械地求解最大公约数

问题的算法。在辗转相除法中分为使用除法运算和使用减法运算两种方法。使用减法运算简单易懂，步骤如图 5.2 所示。用两个数中较大的数减去较小的数（步骤），反复进行上述步骤，直到两个数的值相等（步骤的终止）。如果最终这两个数相同，那么这个数就是最大公约数。请诸位注意以下三点：1. 步骤是明确的、完全不依赖直觉的；2. 步骤是机械的、不需要动脑筋就能完成的；3. 使步骤终止的原因是明确的。




步骤1：42 – 12 = 30，将30写到42的下面
（从较大的数中减去较小的数，将结果写到较大数的下方）
步骤2：30 – 12 = 18，将18写到30的下面
（从较大的数中减去较小的数，将结果写到较大数的下方）
步骤3：18 – 12 = 6，将6写到18的下面
（从较大的数中减去较小的数，将结果写到较大数的下方）
步骤4：12 – 6 = 6，将6写到12的下面
（从较大的数中减去较小的数，将结果写到较大数的下方）
步骤5：两个的值相等了，这个数就是最大公约数

图 5.2　根据辗转相除法求解最大公约数的方法

使用辗转相除法求解 12 和 42 的最大公约数的程序代码如代码清单 5.1 所示。本章展示的程序都是用 VBScript 编程语言编写的。只要把代码清单 5.1 中的内容输入到文本编辑器中，保存成扩展名为 .vbs 的文件，比如 Sample1.vbs，双击这个文件程序就可以运行了（如图 5.3 所示）。诸位即使读不懂这段程序代码的内容也没有关系，这里需要诸位注意的是该算法所描述的步骤是可以直接转换成程序的。

**代码清单 5.1　求解 12 和 42 最大公约数的程序**

```
a = 12
b = 42
While a <> b
    If a > b Then
        a = a - b
```

```
    Else
        b = b - a
    End If
Wend
MsgBox "最大公约数为" & CStr(b) & "。"
```


图 5.3 代码清单 5.1 的执行结果

## 5.4 要点 3：了解并应用典型算法

笔者建议从事编程工作的人手中要有一本能作为算法辞典的书[①]。就像新入职的员工为了书写商务文书去买“商务文书范文”方面的书一样。虽然算法应该由诸位自己思考，但是如果遇到了不知道从哪里下手才好的问题，也可以利用这类辞典查查已经发明出来的算法。

作为程序员的修养，表 5.1 中列出了笔者认为最低限度应该了解的典型算法。这些算法包括刚刚介绍过的求解最大公约数的“辗转相除法”，判定素数的“埃拉托斯特尼筛法”（将在后面介绍），检索数据的三种算法以及排列数据的两种算法。记住了这些典型算法固然好，但是请诸位注意绝不要丢掉自己思考算法的习惯。

① 可以作为算法辞典使用的书有《算法技术手册》（George T. Heineman、Gary Pollice、Stanley Selkow（著），杨晨、李明（译），机械工业出版社，2010 年 3 月）、《算法精解：C 语言描述》（Kyle Loudon（著），肖翔、陈舸（译），机械工业出版社，2012 年 9 月）等。

表 5.1　主要的典型算法

| 名称 | 用途 |
| --- | --- |
| 辗转相除法 | 求解最大公约数 |
| 埃拉托斯特尼筛法 | 判定素数 |
| 顺序查找 | 检索数据 |
| 二分查找 | 检索数据 |
| 哈希查找 | 检索数据 |
| 冒泡排序 | 数据排序 |
| 快速排序 | 数据排序 |

再试着思考一个具体问题吧。这次请思考一下解决“求解 12 和 42 的最小公倍数”这个问题的算法。所谓最小公倍数就是指两个整数的公共倍数（是一个数几倍的数）中最小的那个数。最小公倍数的求解方法诸位在中学的数学课上也应该学过了，但是很可惜求解步骤是依赖人类的直觉的。请再思考一个适用于计算机的机械的算法。诸位说不定会想“反正会有典型算法的吧，比如‘某某氏的某某法’”，然后就纠结于是否还要自己思考。

但是即使查了算法辞典之类的书，也还是找不到求解最小公倍数的算法。为什么呢？因为我们可以通过以下方法求解最小公倍数——用两个整数的乘积除以这两个整数的最大公约数。因此 12 和 42 的最小公倍数就是 $12 \times 42 \div 6 = 84$ 了。如此简单的算法不能算作典型算法。这个例子说明先自己思考算法，再去应用典型算法这一点很重要。

## 5.5　要点 4：利用计算机的处理速度

这次再请诸位思考求解“判定 91 是否是素数”这一问题的算法。在用于判定素数的典型算法中，有一个被称为“埃拉托斯特尼筛法”的算法。在学习这个算法之前，先请诸位思考如果是在数学考试中碰到

了这道题，要如何解答呢？

也许有人会这样想：用 91 分别除以比它小的所有正整数，如果没有找到能够整除的数，那么 91 就是素数。但是，如此繁琐的步骤可行吗？实际上这就是正确答案。埃拉托斯特尼筛法是一种用于把某个范围内的所有素数都筛选出来的算法，比如筛选 100 以内的所有素数，其基本思路就是用待判定的数除以比它小的所有正整数。例如要判定 91 是否是素数，只要分别除以 2～90 之间的每个数就可以了（因为 1 肯定能够整除任何数，所以从 2 开始检测）。这个步骤用程序表示的话，就变成了如代码清单 5.2 所示的代码。Mod 是用于求除法运算中余数的运算符。如果余数为 0 则表示可以整除，因此也就知道待判定的数不是素数了。程序执行结果如图 5.4 所示。

**代码清单 5.2　判定是否是素数的程序**

```
a = 91
s = "是素数。"
For i = 2 to (a - 1)
    If a Mod i = 0 Then
        s = "不是素数。"
        Exit For
    End If
Next
MsgBox CStr(a) & s
```


图 5.4　代码清单 5.2 的执行结果

无论是多么冗长繁琐的步骤，只要明确并且机械就能构成优秀的算法。诸位把算法用程序表示出来让计算机去执行，而计算机会用令人吃惊的速度为我们执行。为了判定 91 是否是素数，用 91 除以 2～90 这 89 个数的操作一瞬间就可以完成。在思考算法时不防时刻记着，解决问题时是可以利用计算机的处理速度的。

作为利用计算机的处理速度解决问题的另一个例子，请试着求解以下联立方程组。题目是鸡兔同笼问题：鸡和兔子共计 10 只，把它们的脚加起来共计 32 只，问鸡和兔子分别有多少只？设有 x 只鸡，y 只兔子，那么就可以列出如下的联立方程组。

$$\begin{cases} x+y=10 & \cdots\cdots\text{鸡和兔子共计 10 只} \\ 2x+4y=32 & \cdots\cdots\text{脚加起来共计 32 只} \end{cases}$$

因为鸡和兔子的只数应该都在 0～10 这个范围内，所以就试着把 0～10 中的每个数依次代入 x 和 y，只要能够找到使这两个方程同时成立的数值也就求出了答案。利用计算机的处理速度，答案一瞬间就出来了（如代码清单 5.3 和图 5.5 所示）。

**代码清单 5.3　求解鸡兔同笼问题的程序**

```
For x = 0 To 10
    For y = 0 To 10
        a = x + y
        b = 2 * x + 4 * y
        If (a = 10) And (b = 32) Then
            MsgBox "鸡 = " & CStr(x) & ", 兔子 = " & CStr(y)
        End If
    Next
Next
```


图 5.5 代码清单 5.2 的执行结果

## 5.6 要点 5：使用编程技巧提升程序执行速度

解决一个问题的算法未必只有一种。在考量用于解决同一个问题的多种算法的优劣时，可以认为转化为程序后，执行时间较短的算法更为优秀。虽然计算机的处理速度快得惊人，但是当处理的数据数值巨大或是数量繁多时还是要花费大量的时间。例如，判定 91 是否是素数的过程一下子就有结果了，可是要去判定 999999937 的话，笔者的电脑就要花费大约 55 分钟之久（言外之意 999999937 是素数）。

有时稍微往算法中加入一些技巧，就能大幅度地缩短处理时间。在判定素数上，原先的过程是用待判定的数除以比它小的所有正整数，只要在此之上加入一些技巧，改成用待判定的数除以比它的 1/2 小的所有数，处理时间就会缩短。之所以改成这样是因为没有必要去除以比它的 1/2 还大的数[①]。通过这一点改进，除法运算的处理时间就能够缩短 1/2。

在算法技巧中有个著名的技巧叫作“哨兵”。这个技巧多用在线性搜索（从若干个数据中查找目标数据）等算法中。线性搜索的基本过程是将若干个数据从头到尾，依次逐个比对，直到找到目标数据。

① 要是从 2 开始依次除下去，只需要从 2 除到待判定数的平方根就足够了。

下面还是通过例题来思考吧。假设有 100 个箱子，里面分别装有一个写有任意数字的纸条，箱子上面标有 1～100 的序号。现在要从这 100 个箱子当中查找是否有箱子装有写着要查找数字的纸条。

首先看看不使用哨兵的方法。从第一个箱子开始依次检查每个箱子中的纸条。每检查完一个纸条，还要再检查箱子的编号（用变量 N 表示），并进一步确认编号是否已超过最后一个编号了。这个过程用流程图表示后如图 5.6 所示。

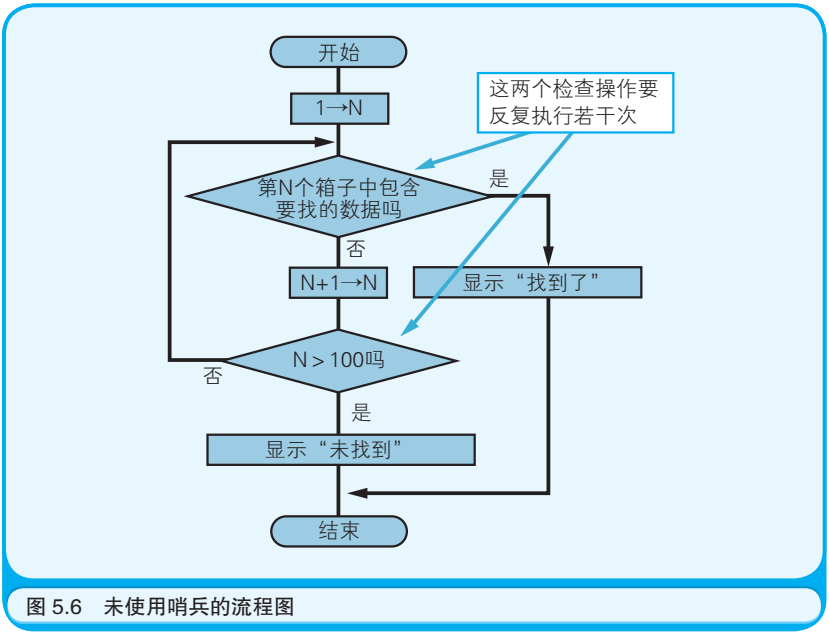



图 5.6　未使用哨兵的流程图

图 5.6 所示的过程，虽然看起来似乎没什么问题，但是实际上含有不必要的处理——每回都要检查箱子的编号有没有到 100。

为了消除这种不必要的处理，于是添加了一个 101 号箱子，其中

预先放入的纸条上写有正要查找的数字。这种数据就被称为“哨兵”。通过放入哨兵，就一定能找到要找的数据了。找到要找的数据后，如果该箱子的编号还没有到 101 就意味着找到了实际的数据；如果该箱子的编号是 101，则意味着找到的是哨兵，而没有找到实际的数据。使用了哨兵的流程图如图 5.7 所示。需要多次反复检查的就只剩下“第 N 个箱子中包含要找的数字吗?”这一点了，程序的执行时间也因此大幅度地缩减了。

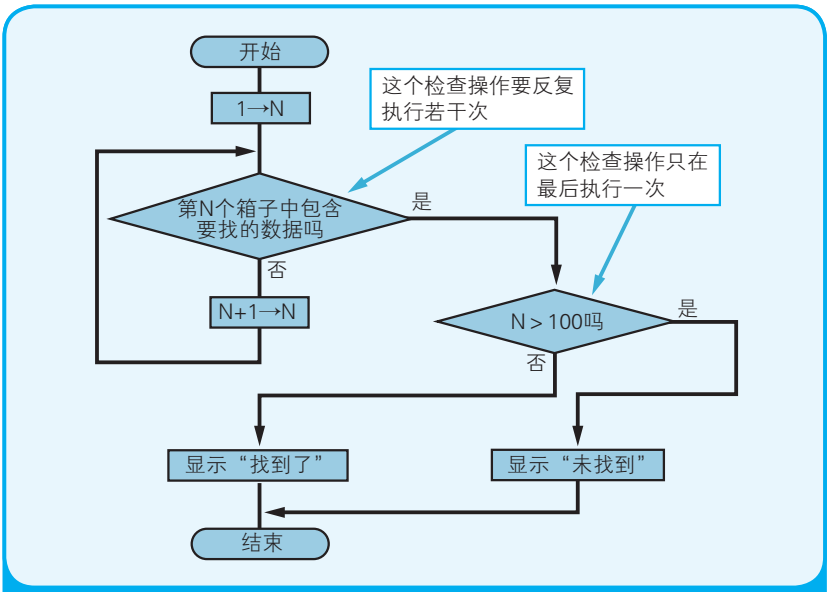



图 5.7 使用了哨兵的流程图

当笔者第一次得知哨兵的作用时，对其巧妙性感到惊叹，兴奋异常。有些读者会感到“不太明白巧妙在哪里”，那么就讲一个故事来解释哨兵的概念吧。假设某个漆黑的夜晚，诸位在在海岸的悬崖边上玩一个游戏（请勿亲身尝试）。诸位站在距悬崖边缘 100 米的地方，地上