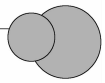
第3章

基本算法和策略



本章的学习目标：

- 什么是基本算法？
- 哪些算法在计算问题求解中最为常用？
- 算法与算法策略有何区别？
- 哪些基本的算法策略在各种算法解决方案中被普遍采用？

本章讲述使用计算机进行问题求解的基本算法，这些算法思想比较浅显，容易理解，所需要的数据结构也最为简单，适于作为算法学习的入门练习。基本算法的介绍中包括问题的陈述和抽象，数据的表达和算法分析，并使用 RAPTOR 描述所有的算法流程。

而算法策略，例如本章介绍的贪心策略和回溯策略等，在许多文献中也被称为算法。这些算法策略体现了计算机科学发展中沉淀下来的智慧与一般性问题处理的优化原则，所以，本章专门辟出一节进行讨论。

3.1 基本算法

基本算法是计算机科学中最为浅显和常用的算法，如蛮力法、递推法和递归法等，还有一些不见得能称得上算法，但却是非常基础的计算案例，例如字符和字符串的应用、模运算等，这对于初学者开阔眼界，掌握基本的算法工具也是十分重要的。

3.1.1 蛮力法

进行计算机问题求解的第一号方法被称为蛮力法(brute force)，也称穷举法，是一种简单而直接的问题求解方法，解题思路常常直接基于问题的描述，因此，也是最容易应用的方法。但是，用蛮力法设计的算法其时间性能往往是最低的，典型的指数时间算法一般都是通过蛮力搜索而得到的。

采用蛮力算法解题的基本思路如下：

(1) 确定穷举对象、穷举范围和判定条件。

(2) 一一穷举可能的解，验证是否是问题的解。

蛮力法所依赖的最基本技术是扫描技术，依次处理所有元素是蛮力法的关键，依次处理每个元素的方法是遍历。也就是指从可能的集合中一一枚举各个元素，用题目给定的

约束条件判定哪些是无用的，哪些是有用的。能使命题成立者即为问题的解。在蛮力算法中，穷举对象的选择也是非常重要的，它直接影响着算法的时间复杂度，选择适当的穷举对象可以获得更高的效率。下面就从穷举对象的选择以及判定条件的确定这两个方面来探讨如何使用蛮力法解题。

例 3-1 百钱买百鸡问题：某个人有一百块钱，打算买一百只鸡。公鸡五块钱一只，母鸡三块钱一只，小鸡一块钱三只(参见图 3-1)。请编一个算法，算出如何能刚好用一百块钱买一百只鸡？

图 3-1 百鸡图(局部)

解：此题可以试用蛮力法来解，以 3 种鸡的个数为穷举对象(分别设为 x、y 和 z)，以 3 种鸡的总数($x+y+z$)和买鸡用去的钱的总数($5x+3y+z/3$)为判定条件，穷举各种鸡的个数。由于 3 种鸡的和是固定的，只要穷举两种鸡(x 和 y)，第 3 种鸡就可以根据约束条件求得($z=100-x-y$)，这样就缩小了穷举范围，该算法的结果和流程参见图 3-2。

从上例可以看出，对于蛮力算法，加强约束条件，缩小穷举的范围，是算法流程优化的主要考虑方向。

虽然高效的算法很少使用蛮力法，但是，基于以下原因，蛮力法也是一种重要的算法设计技术：

(1) 理论上，蛮力法可以解决可计算领域的各种问题。对于一些非常基本的问题，例如求一个序列的最大元素，计算 n 个数的和等。所以，蛮力法是一种很常用的算法设计技术。

(2) 蛮力法经常用来解决一些规模较小的问题。如果需要解决的问题规模不大，用蛮力法设计的算法其速度是可以接受的，此时，设计一个更高效算法的代价是不值得的。

(3) 对于一些重要的问题(例如排序、查找和字符串匹配)，蛮力法可以产生一些合理的算法，因此具备一些实用价值，而且不受问题规模的限制。

(4) 蛮力法可以作为某类问题求解的时间性能的底限，来衡量同样问题的更高效算法。

3.1.2 分段函数

收费问题与我们的生活息息相关，如水费问题、电费问题和话费问题等，这些收费问题往往根据不同的用量采用不同的收费方式。以收费为内容的数学问题多以分段函数的形式出现。

例 3-2 为鼓励市民节约用电，某市开始采取按月用电量分段收费办法，某户居民每月应交电费 y(元)与用电量 x(度)的函数如下：

$$y=\begin{cases}0.65x, & 0\leqslant x\leqslant 100\\ 0.8x-15, & x>100\end{cases}$$

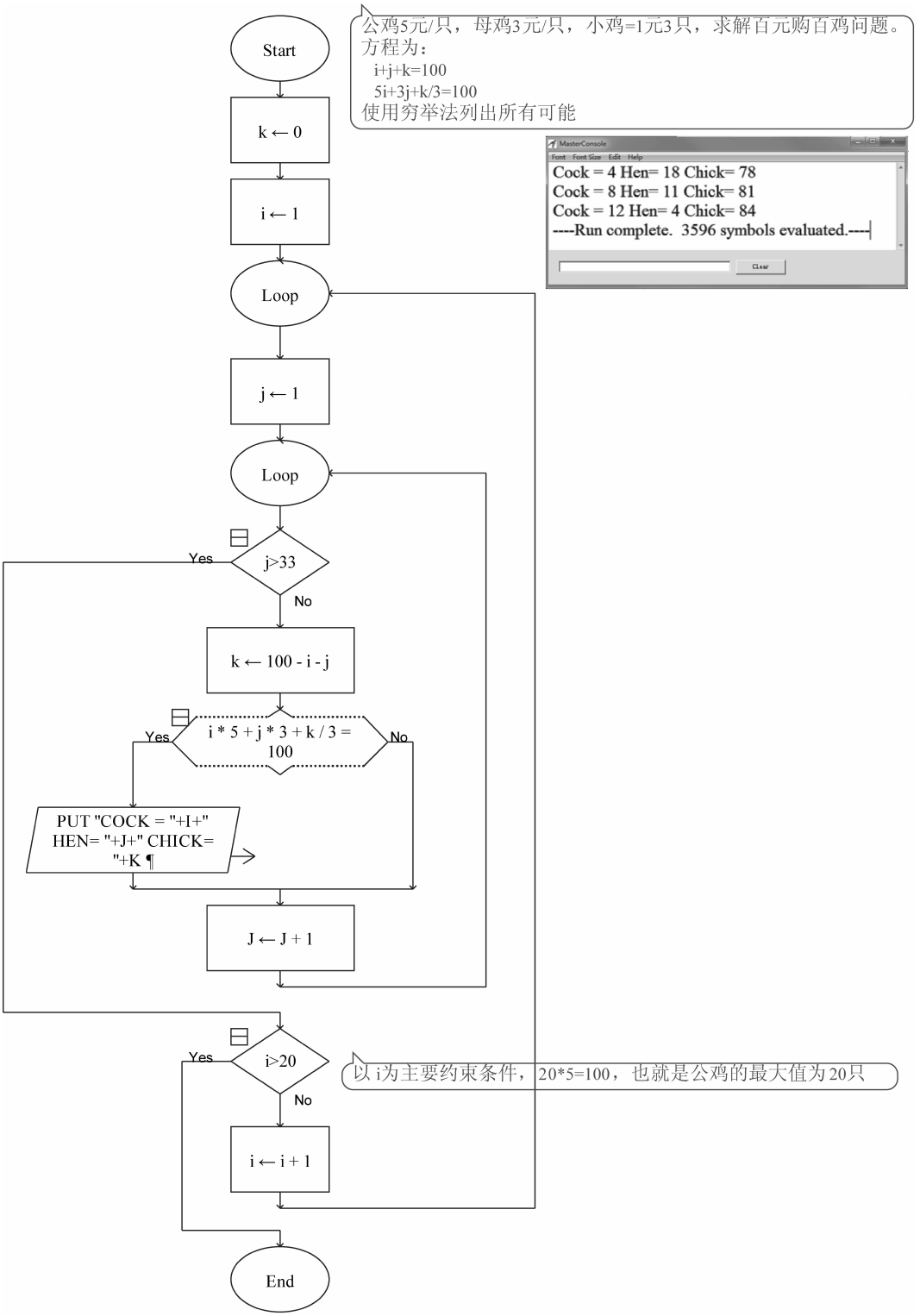


图 3-2　百元百鸡问题的算法结果和流程图

请设计上述电费的收费算法。

解：该分段函数题的算法看起来似乎很简单，主要是判断输入的数据是否在分段的临界点上，然后得出计算结果，其算法如图 3-3 所示。

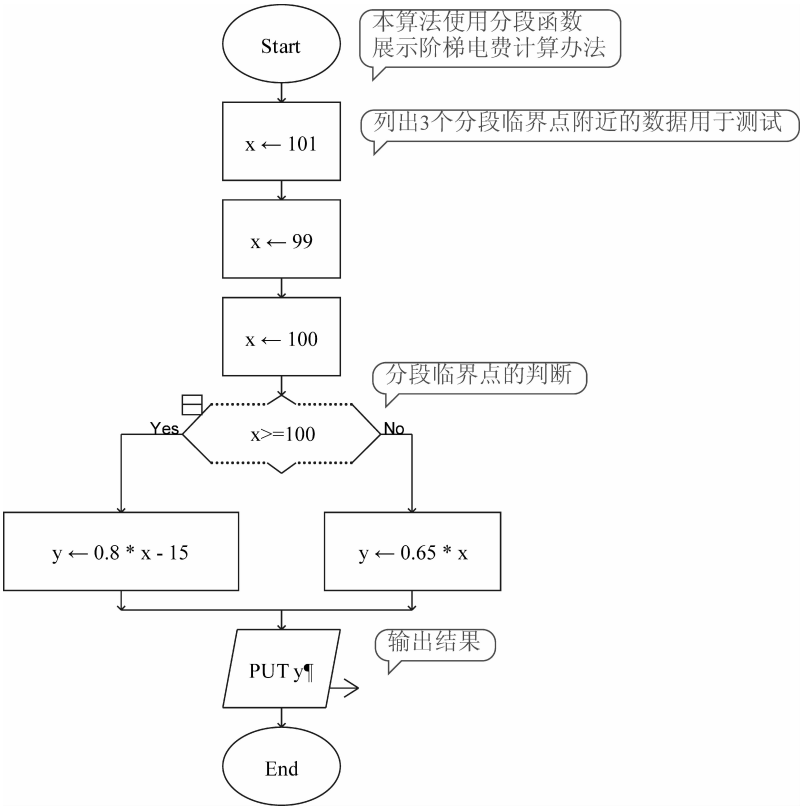


图 3-3　分段函数的计算案例

但是在事实上，这个算法存在或者可能产生诸多问题：

（1）对于初学者来说，最容易犯的错误是将函数的数学表达式直接搬到算法中，例如：0<＝x<100。这样的表达式，RAPTOR 并不认为有错误。当然，计算结果也就会不合逻辑。

（2）例 3-2 的函数定义中，没有定义 $x<0$，也就是输入为负数的时候如何处理。当然，读者可以实验一下，当 x 取负值时结果正确吗？

（3）这个算法没有设计输入和规范的输出界面，如果是完整的收费算法，必须定义输入界面，并且要提示用户输入的数值（包括量纲，例如，电费算法的输入应该是以度或 kWh 为单位）。而正式的输出界面中，必须包含输出数字和有关的量纲（例如，元或 Yuan）。

3.1.3　递推法

递推法是利用问题本身所具有的一种递推关系求解问题的一种方法。所谓递推，是

指从已知的初始条件出发，依据某种推算关系，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后确定。

设要求问题规模为 N 的解，当 $N=1$ 时，解或为已知，或能非常方便地得到解。能采用递推法构造算法的问题有重要的递推性质，即当得到问题规模为 $i-1$ 的解后，由问题的递推性质，能从已求得的规模为 $1,2,\cdots,i-1$ 的一系列解，构造出问题规模为 i 的解。这样，程序可从 $i=0$ 或 $i=1$ 出发，重复地，由已知 $i-1$ 规模的解，通过递推，获得规模为 i 的解，直至得到规模为 N 的解。

可用递推算法求解的题目一般有以下两个特点：

(1) 问题可以划分成多个状态。

(2) 除初始状态外，其他各个状态都可以用固定的递推关系式来表示。

在实际解题中，题目不会直接给出递推关系式，而是需要通过分析各种状态找出递推关系式。

例 3-3　猴子吃桃问题。有一天小猴子摘了若干个桃子，立即吃了一半，还觉得不过瘾，又多吃了一个；第二天接着吃剩下桃子的一半，仍觉得不过瘾，又多吃了一个；以后小猴子都是吃剩下的桃子一半多一个。到第 10 天小猴子再去吃桃子的时候，看到只剩下一个桃子，则小猴子第一天共摘了多少桃子？

解：由题意可以得到表 3-1。

表 3-1　猴吃桃问题的状态递推

天数	1	2	3	4	5	6	7	8	9	10
桃数	1534	766	382	190	94	46	22	10	4	1

分析后可知，猴子吃桃问题的递推关系为

$$S_n=\begin{cases}1, & (n=10)\\ 2(S_{n+1}+1), & (1\leqslant n<10)\end{cases}$$

在此基础上，以第 10 天的桃数作为基数，用以上归纳出来的递推关系设计出一个循环过程，将第 1 天的桃数推算出来。猴子吃桃问题的递推算法参见图 3-4。

算法说明如下：

(1) 为了加强交互性，增加了交互界面，可以输入不同的天数进行递推。

(2) 算法使用了两个变量，一个用于递推的循环控制，另一个保存递推得到的结果。

(3) 注意主要的循环控制变量是递减的，与题意设计完全相同，便于理解。

(4) 算法中加入了输入数据的正确性控制，也就是不可以输入 0 和负数。

递推算法在许多数列的计算中也有应用，例如著名的 Fibonacci 数列等，都可以使用递推法求解。

例 3-4　阶乘计算。编写算法，对给定的 $n(n\leqslant 100)$，计算并输出 k 的阶乘 $k!(k=1,2,\cdots,n)$的全部有效数字。

解：由于要求的整数可能大大超出一般整数的位数，算法用一维数组存储长整数，数组的每个元素只存储长整数的一位数字。如有 m 位长的整数 N 用数组 $a[]$存储：

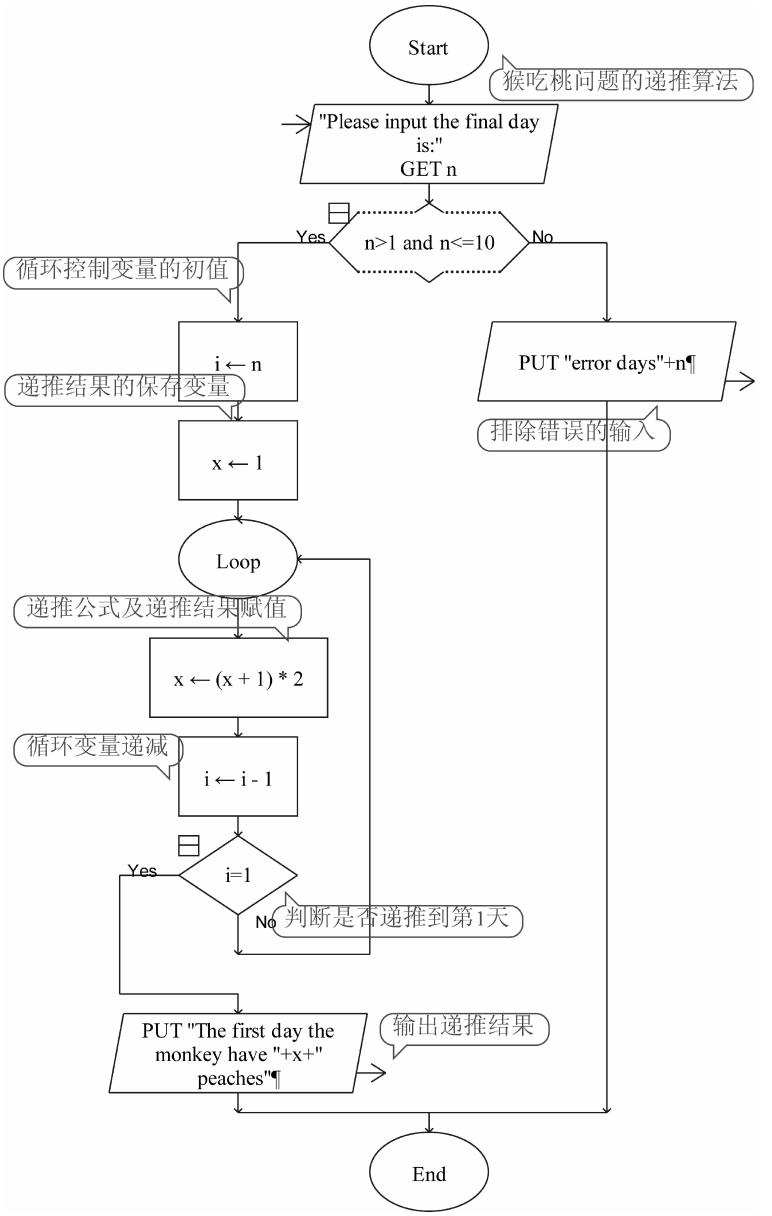


图 3-4　猴子吃桃问题的递推算法

$$N=a[m]\times 10^{m-1}+a[m-1]\times 10^{m-2}+\cdots+a[2]\times 10^{1}+a[1]\times 10^{0}$$

并用 length_of(a)存储长整数 N 的位数 m，即 length_of(a)＝m。按上述约定，数组的每个元素存储 k 的阶乘 $k!$的一位数字，并从低位到高位依次存于数组的第二个元素、第三个元素、…。例如，5!＝120，在数组中的存储形式为：

a

3	0	2	1

首元素 3 表示长整数是一个 3 位数，接着是低位到高位依次是 0、2、1，表示整数 120。

计算阶乘 $k!$ 可采用对已求得的阶乘 $(k-1)!$ 连续累加 $k-1$ 次后求得。例如，已知 $4!=24$，计算 $5!$，可对原来的 24 累加 4 次 24 后得到 120。

由于安全需要，目前主流的公钥加密算法（如 RSA[①]）都是基于二进制的 512 位或 1024 位的大整数，而目前的主流高级语言编译器最多也只能支持到二进制 64 位整数，所以大整数的存储和运算对于 RSA 算法的实现也是至关重要的。一个最容易理解的方法就是将大数用十进制表示，并将每一位（0～9）都作为一个单独的数字作为数组元素来用数组进行管理。做加减乘除等运算时，通过设计算法对其进行进位和借位。这也是大部分程序设计环境对超出处理器（CPU）字长处理能力的大数运算的处理方法。

具体的算法流程，请读者自行编写。

3.1.4　模运算

在算法应用中，有一类计算与模运算（求除法的余数）有关，例如，一个星期的模为 7 天，一天的模是 24 小时或 1440 分钟，一年的模是 12 个月（也可以是 365 或 366 天）。而模运算在数字比较和诸多计算案例中都有应用。

例 3-5　求 100～1000 范围内的平方回数。所谓平方回数，是指某个数既是一个回文数，又是一个平方数。例如，121 是回文数，又是 11 的平方。

解：一个 100 以上的平方数，必须从平方根大于等于 10 的数字开始计算，而 1000 可以作为搜索循环的控制变量，但是，如何求出回文数？由于是在 3 位数中求回文数，也就是要求个位与百位上相等。这个时候，模运算（在 RAPTOR 中模运算符为 mod）就可以派上用场。具体算法参见图 3-5。

例 3-6　求闰年的判定方法。

解：公历闰年判定遵循的规律为：4 年一闰，100 年不闰，400 年再闰。所以，公历闰年的计算方法如下（符合以下条件之一的年份即为闰年）：

（1）能被 4 整除而不能被 100 整除。

（2）能被 400 整除。

闰年的判定方法的算法参见图 3-6。

3.1.5　字符和字符串运算

字符和字符串运算在算法中涉及的地方较多，尤其是在信息网络传播的过程中扮演了重要的角色。归纳起来，字符和字符串运算在算法中的用途有以下几个方面：

（1）在输入输出界面中的应用。例如，在输入过程中提示用户输入变量的值，在输出过程中将计算结果与量纲结合在一起向用户展示。

---

① RSA（公钥加密算法）是 1977 年由 Ron Rivest、Adi Shamirh 和 Len Adleman 在美国麻省理工学院开发的。RSA 取名来自 3 位开发者姓氏的首字母。RSA 是目前最有影响力的公钥加密算法，它能够抵抗到目前为止已知的所有密码攻击，已被国际标准化组织（ISO）推荐为公钥数据加密标准。RSA 算法基于一个十分简单的数论事实：将两个大素数相乘十分容易，但想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

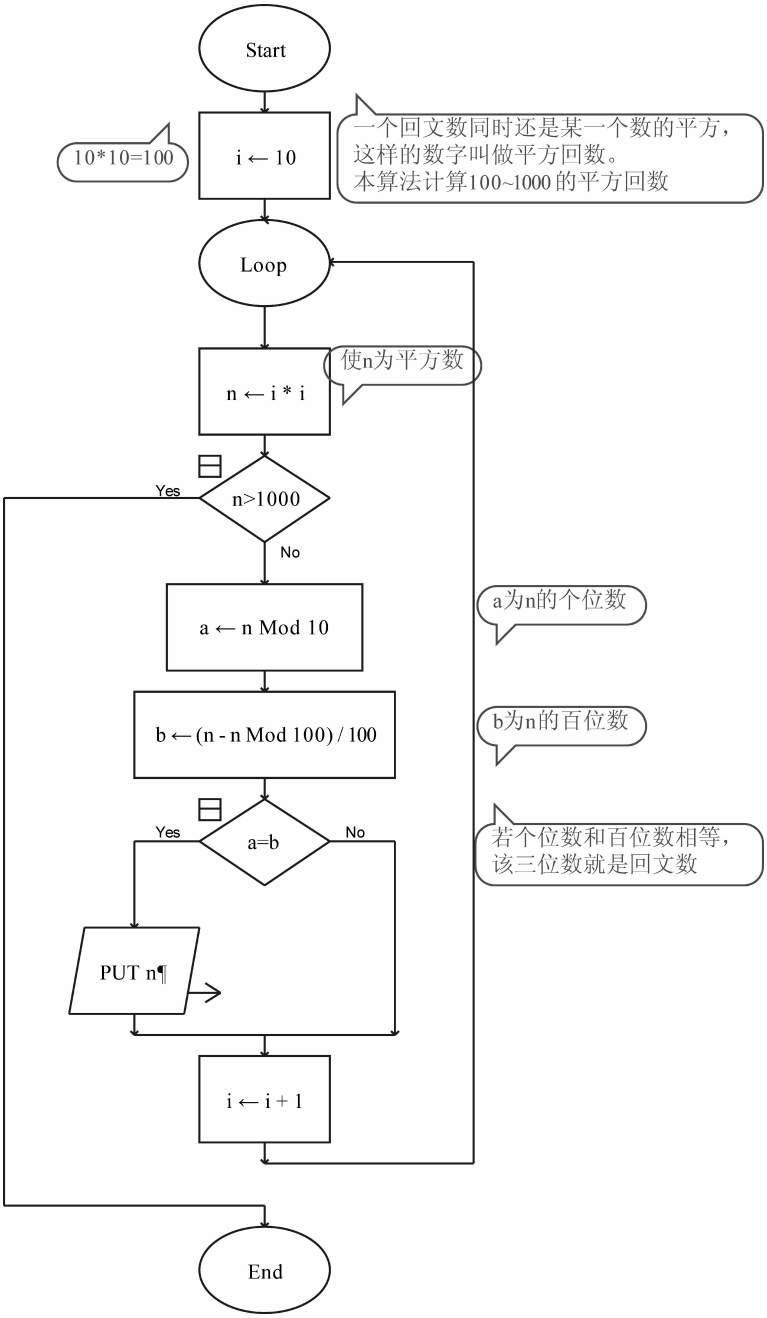


图 3-5　模运算的一个案例：求平方回数

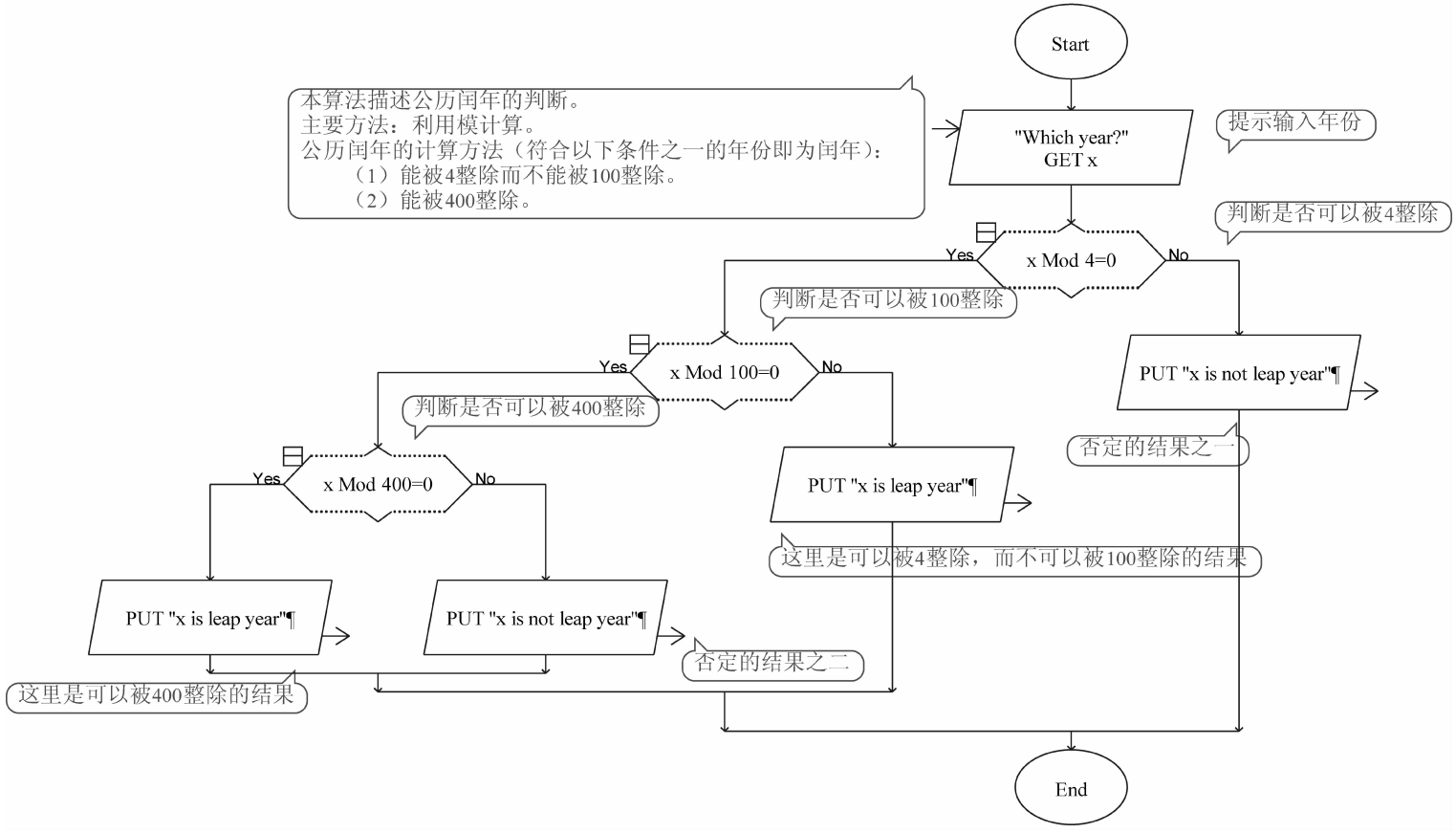


图 3-6　公历闰年判定的算法

(2) 在信息安全中的应用。例如,信息的加密与解密(所有的字符编码使用数字表达,因此可以通过计算进行加解密)。

(3) 对用户在特定应用中输入的字符串进行模式正确性的判断。例如,电子邮件地址需包含@符号等。

值得注意的是,在 RAPTOR 中,尽管帮助文件目前只提到两种变量类型(参见帮助文件中 Variables in RAPTOR):数值(number)和字符串(string)。而实际上,RAPTOR 中还有一种数据类型称为字符(character),与字符串可以用过输入语句和赋值语句赋值不同,字符型数据只能通过 to_character()函数和访问字符串得到。在 RAPTOR 的运行数据显示窗口,读者可以注意到字符串与字符类型数据的区别,字符串的数据是由双引号包含,而字符型数据则使用单引号。由于是不同类型的数据,所以一个重要的问题是:在 RAPTOR 中,这两类数据的操作非常微妙,也就是一个字符串类型的变量可以与一个字符变量(使用+运算符)连接,但不可以进行关系操作(比较大小相等),参见图 3-7。

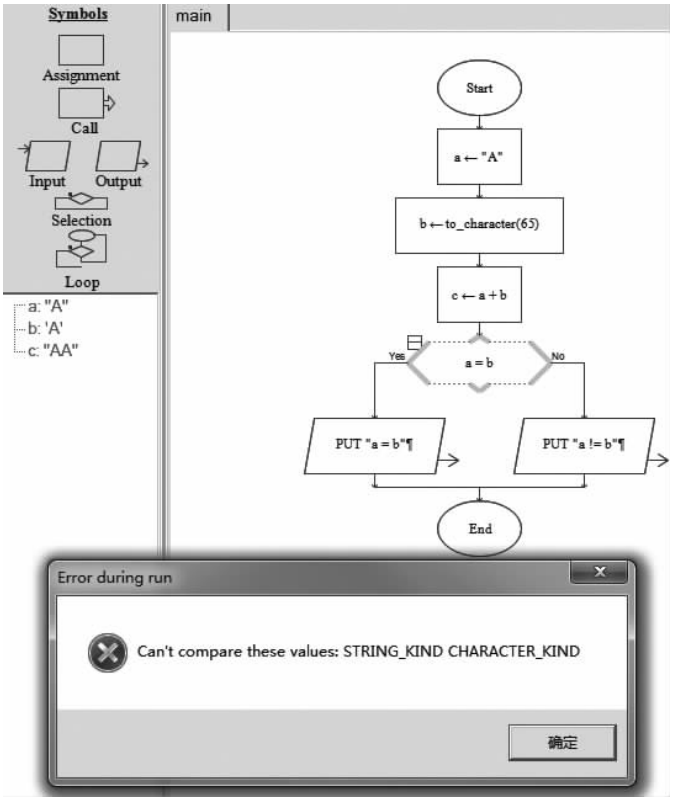


图 3-7 字符串与字符类型数据不能进行关系运算

在 RAPTOR 中,除了 to_character()函数可以把一个数字传换成 ASCII 码字符之外,还有一个逆运算函数 to_ascii(),它是将一个字符串中的某个字符取出,转换成为数字,以便进行编码的性质判断。有了这个函数,就可以进行一个字符属于文字(a～b,A～B)、数字(0～9)、标点符号(,.'";:+- * /()[]{}@!# $ %^&_=|? ～)、空格(space)还是其他控制符号的判断,这就为大量的与符号处理有关的算法提供了基本手段。