第2章　算法运用

第04讲　算法构造

第04讲　算法构造

算法是智能计算领域最核心的概念之一。当你拥有了一台机器，希望机器系统能够为你服务，解决你需要解决的某个智能计算任务，那你首先要给出完成这项任务的算法步骤。比如，你希望具备猜扑克牌颜色这样一个魔术游戏的能力，那么你就需要为机器编制完成这样任务的算法。如果说对于计算机科学技术的学生而言，玩的就是算法，那么对于智能科学技术的学生而言，玩的就是智能算法！

一般而言，所谓算法就是给出解决一个（智能）计算问题具体步骤的集合。我们在前面“机器系统”一讲中已经遇到过一些简单的指令执行算法。比如中央处理机常规处理“算法”就是，只要未发出停机指令就执行以下步骤：（1）获得指令；（2）指令解码；（3）执行指令。

1、界定算法的性质

“煮鸡蛋吃”算法如下：

（1）从冰箱里取一枚鸡蛋；

（2）将鸡蛋放进煮锅；

（3）锅里加水直到盖满鸡蛋；

（4）持续给锅加温直到沸腾为止；

（5）停止加温取出鸡蛋；

（6）将鸡蛋放入凉水浸泡1分钟；

（7）敲破鸡蛋壳，去除全部蛋壳；

（8）将去壳后的鸡蛋一口一口吃掉。

1、界定算法的性质

通过上述算法例子，我们不难了解算法的一些特点。但作为机器系统能够严格精确执行的操作步骤集合，我们必须对算法下一个严格的定义：

算法是一组明确的、可以直接执行之步骤的有限有序集合。

（1）有序性：算法中所有步骤均规定有执行顺序的；

（2）有限性：算法中的步骤是有限的；

（3）明确性：集合中的每条指令均是明确的、可以直接执行的步骤。

有时候，我们还会要求每一个算法不但构成步骤是有限的，而且还要求这些步骤的动态执行也是在有限时间中能够结束的，即所谓（4）终止性。不过，对于特殊算法，有时候我们却需要永不终止，如操作系统。关于这个话题，更多地涉及到计算理论的内容，并跟算法效率的讨论有关。我们不做展开了。

1、界定算法的性质

对于算法而言，还有一个重要的方面就是一定区分算法内涵与算法描述之间的区别。

算法的内涵是指一个算法所固有的功能本质，完成某一任务的具体步骤及其内在联系。

而算法的描述则是具体给出的一种表示方式。一个算法可以有不同的描述文本，这些描述文本完成的任务完全一致，因此代表着一个相同的抽象本质。

就好像一本书与一个故事的区别，一个故事可以写成不同版本的书，而这不同版本的书却讲述的是同一个故事。

算法的抽象本质，是任务固有的复杂本性所决定的，而算法的表示只是完成这一任务之算法的一种具体描述。

如果我们考虑到算法的执行问题，还需要区分程序、算法与进程三者的关系。程序是算法的描述，进程是执行程序的活动，而执行一个程序就是执行这个程序所表达的算法，因此进程是算法执行活动。

1、界定算法的性质

最后，对于算法，还要考虑算法的效率与正确性问题。

效率是指执行一个算法所要花费的时空代价。

时间代价是指算法执行中花消的时间，而空间代价是指算法执行中花消的内存。

当然，一般我们仅仅从理论上来讨论算法的效率，并不考虑具体的机器系统，加上空间代价可以转化为时间代价。因此，在考虑算法的效率时，主要查看对于给定的输入数据规模，一个算法需要动态执行多少个计算步骤，称为算法的计算复杂性。

算法的计算复杂性是由算法所解决的问题本身的复杂本性所决定的，这是算法问题的计算复杂性。不过，同样的问题可以有不同的算法来解决，这些不同算法的计算复杂性才是反映算法效率的关键。我们希望，对于给定的问题，都能够找到最低计算复杂性的那个算法，刚好反映了问题本身的计算复杂性。

1、界定算法的性质

衡量算法计算复杂性一般用输入数据的规模来比照，也即算法计算复杂性是其输入数据规模的函数，通常记为：

O（f（n））

其中n为算法输入数据的规模，f（）为算法的计算复杂性函数，O（）为复杂性当量。

从理论上讲，可以将算法按照计算复杂性函数类别来进行分类，比如函数为对数函数的就称为对数复杂性，多项式的就称多项式复杂性（又分为k次多项式）、指数的称为指数复杂性。

1、界定算法的性质

至于算法的正确性，则是要确保算法确实解决了给定的问题。目前，证明算法正确性的方法主要有两种途径：

一是软件测试途径，就是具体地运行算法的程序，采用各种选择测试数据的方法来系统地测试软件，分析算法的结果是否符合规定的（中间环节或最终结果）输出要求；

另一种是程序正确性证明方法，主要是从理论上分析证明算法的正确性。

当然我们希望所构造的算法都是正确的，即确实刚好解决了所要解决的问题，不多也不少。但实际上，当需要解决的问题足够复杂时，很难保证构造的算法是正确的，往往或存在许多漏洞（bug），常常会给各种智能系统带来灾难性的后果，比如航天飞机的发射失败，往往就是因为软件算法上的一点点小问题导致的。因此，算法设计也是大事，不可不慎！

2、描述算法的伪码

构造算法的前提，首先是要给出某种算法表示的方式。尽管算法功能是超越具体的表达形式的，但功能总是要通过一定的形式来呈现的。日常生活中我们一般采用自然语言来发表我们的思想，也常常采用某图式来表达事物及其关系。但对于算法而言，由于明确性的要求，因此往往需要某种精确的形式语言来作为算法表达的语言，这种可以精确描述算法的形式语言就成为原语。

定义原语一般包括两个方面的考虑，语法和语义。语法规定原语中符号组合的规则，语义则说明原语中符号及其组块的含义。

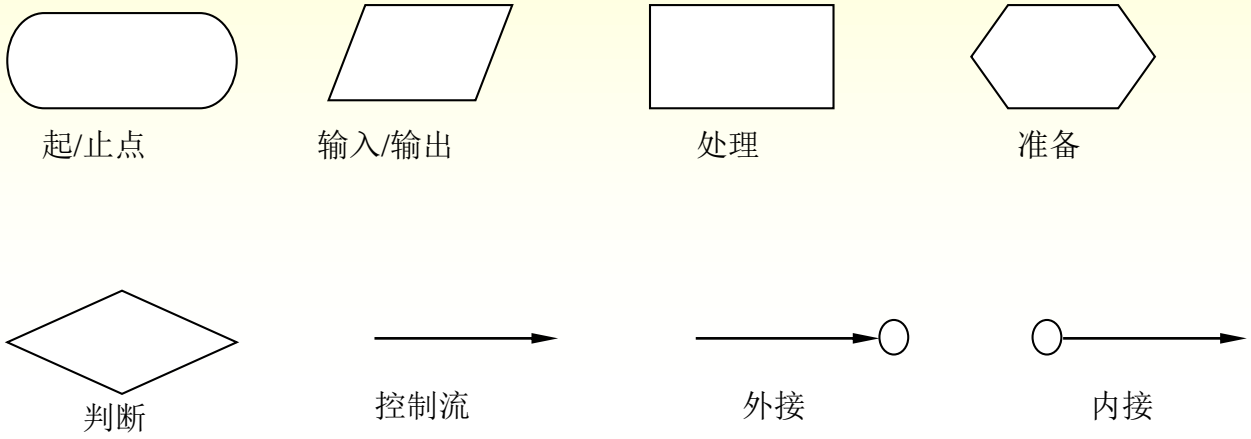
为了既兼顾机器算法执行的精确可行性，又兼顾人们算法描述的直观方便性，并且具有某种通用性，因而可以忽略实现某种程序设计语言的细节，一般采用一种称为伪码的符号系统来作为描述算法的原语。

2、描述算法的伪码

所谓伪码（psudocode）是一种重在表达算法思想的非正式符号系统，常常用在算法的开发进程之中。与一般高级程序设计语言相比较，伪码的主要特点是，既具有直观方便性的优点，又忽略了严格语法的规范性。

由于算法思想主要是通过各种递归语义结构来描述的，因此与所有描述算法的语言符号系统一样，伪码必须要能够给出各种递归语义结构的表达方法。有时为了直观表达算法的思想，也常采用一种称为流程图的图式来对伪码描述的算法作补充说明。

2、描述算法的伪码



流程图的基本符号

上图给出了流程图的基本符号，下面我们针对主要的递归语义结构，结合流程图表示，来给出一种算法原语具体规范。

2、描述算法的伪码

（1）赋值语义结构：如果用name表示变量名称，用expression表示与name有关的数值表达，那么我们称：

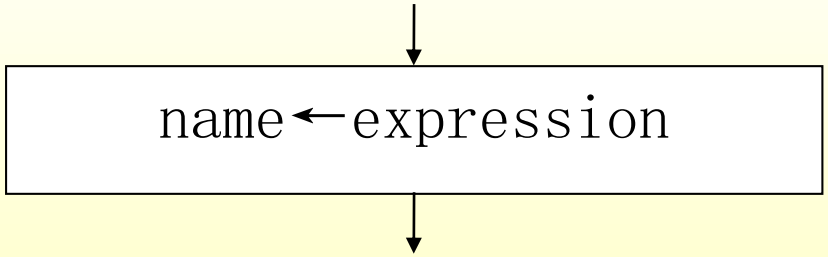
name←expression

为一个赋值语义结构，其含义表示“把expression的值赋给name”。比如：

x←3+4y

就表示将3+4y表达式计算结果的值赋给x。一旦这一计算结束后，变量x的值，就变成“3+4y”了，但注意变量y的值保持不变。

赋值语句流程图



2、描述算法的伪码

（2）条件语义结构：根据某个条件式的是否成立，来决定采取进一步的计算活动。表示这样算法步骤的语义结构，就称为条件语义结构。一般采用如下的描述形式：

if（条件）then（活动1）else（活动2）

或者

if（条件）then（活动）

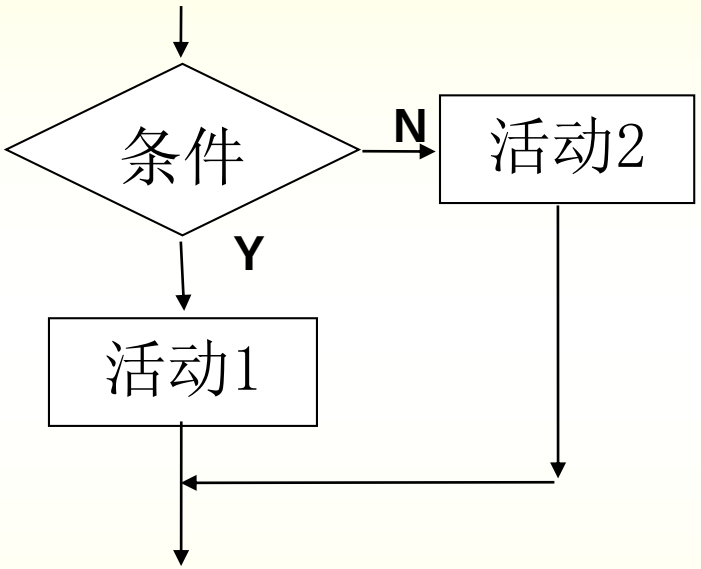
其中if、then、else这些关键词称为保留词（不允许算法设计者用做命名变量名称的词，称为保留词），用于界定一个条件语义结构不同部分（内部）和作用范围（外部）。

2、描述算法的伪码

在上述条件语义结构中，第一个式子的含义表示“如果（条件）成立，就执行（活动1），否则执行（活动2）”，第二个式子的含义表示“如果（条件）成立，就执行（活动）”。比如，

if（x≤6）**then**（y←x+2）**else**（y←x+6）

就是一个条件语义结构。



条件语句流程图

2、描述算法的伪码

（3）循环语义结构：用来表示只要某个条件式保持为真就继续规定的计算活动，表示这样的算法步骤的语义结构，就称为循环语义结构。一般采用如下的描述形式：

while（条件）do（活动）

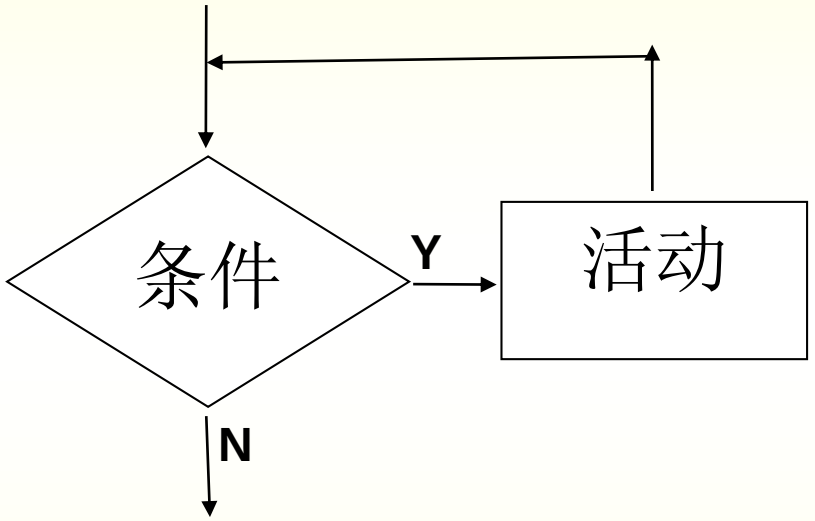
其中while和do 也是保留词。循环语义结构的含义是“检查（条件）是否满足，如果其为真就执行（活动），并返回再次检查（条件）。只有当某次检查（条件）不满足时，才结束算法步骤。”

第3.2节　算法构造

比如x初始值假定为0，那么

while（x≤5）do（x←x+1）

就意味着一直执行6次（x←x+1）计算，结果x的值变成了6。



while语句流程图

2、描述算法的伪码

有了上述三种基本的语义结构描述形式，就可以用来表示完整的算法思想了。

通常一个算法中的每一个相对独立的步骤，称为一个语句，均可以用上面三种语义结构之一来描述，称为具体应用，

而这些语义结构具体应用的语句有序集合，就称为算法的伪码表示。

2、描述算法的伪码

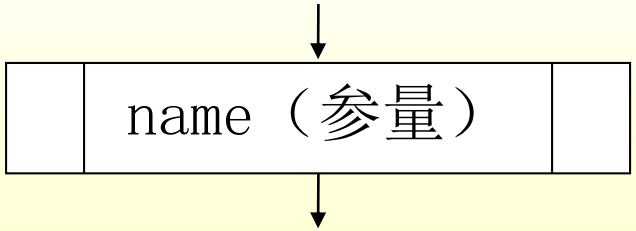
进一步，对于重复出现的伪码段或者相对独立的一段代码，可以用固定名称加以命名定义，称为过程，然后在需要出现的地方直接用该名称替代这段伪玛。一般过程的定义方式为

procedure name（参量）

伪玛段

引用之处直接用语句“**procedure** name”来替代所定义的这段“伪玛段”。

过程语句流程图



2、描述算法的伪码

比如下面就定义了一个称为greetings的过程：

```
procedure greetings（y）
x←y;
while（x≤6）do
   （print（“hello”）；
     x←x+1）
```

此时，在其他需要完成这一过程功能的地方，只需要直接调用这一过程名即可，比如

```
if（x≤10）then（procedure greetings（3））
```

2、描述算法的伪码

有时候，为了使得伪码可读性更高，在嵌套的语句中，每一层语句的结束，都用明显的标识来醒目地加以标记，比如**end while**、**end if** 等，使得伪码的嵌套层次更加一目了然。当然，这种做法并非是强制性的，依赖于个人偏好。作为中国人，有时也可以用汉语保留词来替换英语保留词，以及只要不影响算法思想的表达，可以采用你自己认为的习惯方式来规定你的伪码表达习惯，前提是要让别人能够理解可读。

3、算法构造的过程

有了表示算法的伪码原语，现在我们可以来介绍如何编写解决具体问题的算法了，即所谓的算法构造。

针对某个具体需要解决的问题，算法构造可以分为两个阶段：

发现解决该问题的算法，

以及用伪码将发现的算法表达出来。

如果你已经熟练掌握了伪码的使用，那么很显然，构造算法的重点在于发现算法。其实，发现算法就是一个理解解决问题的过程。

3、算法构造的过程

从算法发现的角度看，可以将解决问题的一般原理对应到如下这样四个阶段上：

阶段1　理解问题

阶段2　寻找一个可能解决问题的算法过程（思路）

阶段3　阐明算法并且用程序将其表达出来

阶段4　从准确度以及作为解决其他问题的一个工具的潜力这两个方面来评估这个程序。

3、算法构造的过程

具体解决问题的思路有许多，比如正向思维、逆向思维（均举扑克牌游戏）、混合思维以及灵机一动等等。

但对于算法发现而言，主要的难点不在于去解决一个问题的特定实例，而是要找出一种适合一个问题所有实例的一般算法。比如要给出加法运算的一般算法，而不是仅仅解决1+1这一个加法运算的特定实例。

这时，你就会发现，对于给定问题，发现其解决算法并非是一个容易的问题。甚至对于许多问题，根本就不存在可以加以解决的算法，比如整数方程解的问题。

3、算法构造的过程

举例1：计算1+2+...+100（n）

0+1=1
1+2=3
3+3=6
6+4=10
10+5=1
5...
4851+99=4950
4950+100=5050

a. 重复地进行相加运算
b. 本次的和作为下一次相加运算的被加数
c. 加数有规律地变化着

a. 令s表示被加数（初始值为0），令I表示加数（初始值为1）
b. 进行100次加法后结束，或者当加数大于100时结束
c. S中存放计算结果

3、算法构造的过程

举例1：计算1+2+...+100

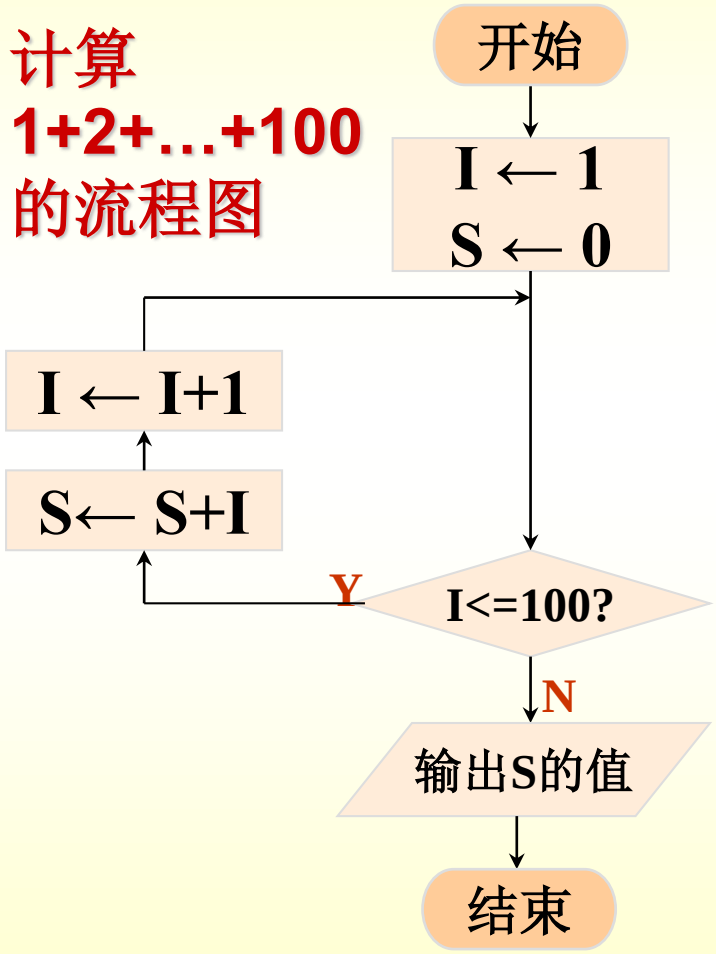
- 下面的算法中S表示被加数，也表示累加和，I表示加数

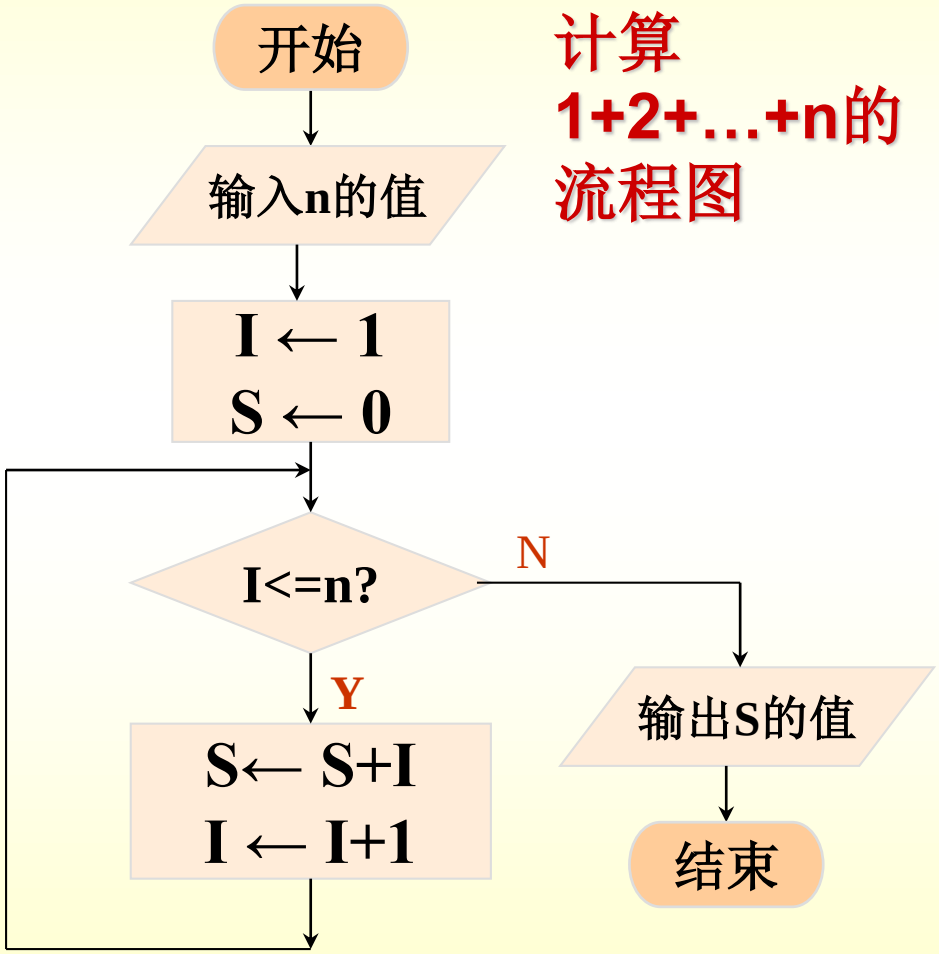
步骤1：S ← 0；I ← 1
步骤2：若I小于等于100，则转向步骤3；否则，转向步骤6；
步骤3：S ← S + I；
步骤4：I ← I + 1；
步骤5：转向步骤2；
步骤6：S的值就是计算结果，算法结束。

推广该算法，即修改算法中步骤2的100为n，可得到计算1+2+...+n的算法。

3、算法构造的过程

计算
1+2+…+100
的流程图





计算
1+2+…+n的
流程图

3、算法构造的过程

举例1：计算1+2+...+n

构造算法另一种思想：例如，计算1+2+...+100时先计算1＋100，2+99，...，49+52，50+51，然后用101×50即可得到运算结果。类推，要计算1+2+...+n，只要知道有多少个n+1，就可以得到计算结果。

步骤1：S ← n+1

步骤2：若n是偶数，则 S ← (S × n) ÷2;

否则， S ← S× (n-1) ÷2 + (n + 1) ÷2

步骤3：S的值就是计算结果，算法结束。

同一个问题可用不同的方法解决，即用不同算法解决同一个问题。因此，就存在一个算法的比较(分析)和选择问题，比较的依据是算法的效率。

3、算法构造的过程

举例2：判断闰年问题，给定一个年号，判断是否是闰年。

问题分析：能被4整除，但是不能被100整除的年份是闰年；能同时被100和400整除的年份是闰年。

步骤1：令y表示年份，给y一个年份值；

步骤2：若y能被4整除并且不能被100整除，则转向步骤5；否则，转向步骤3；

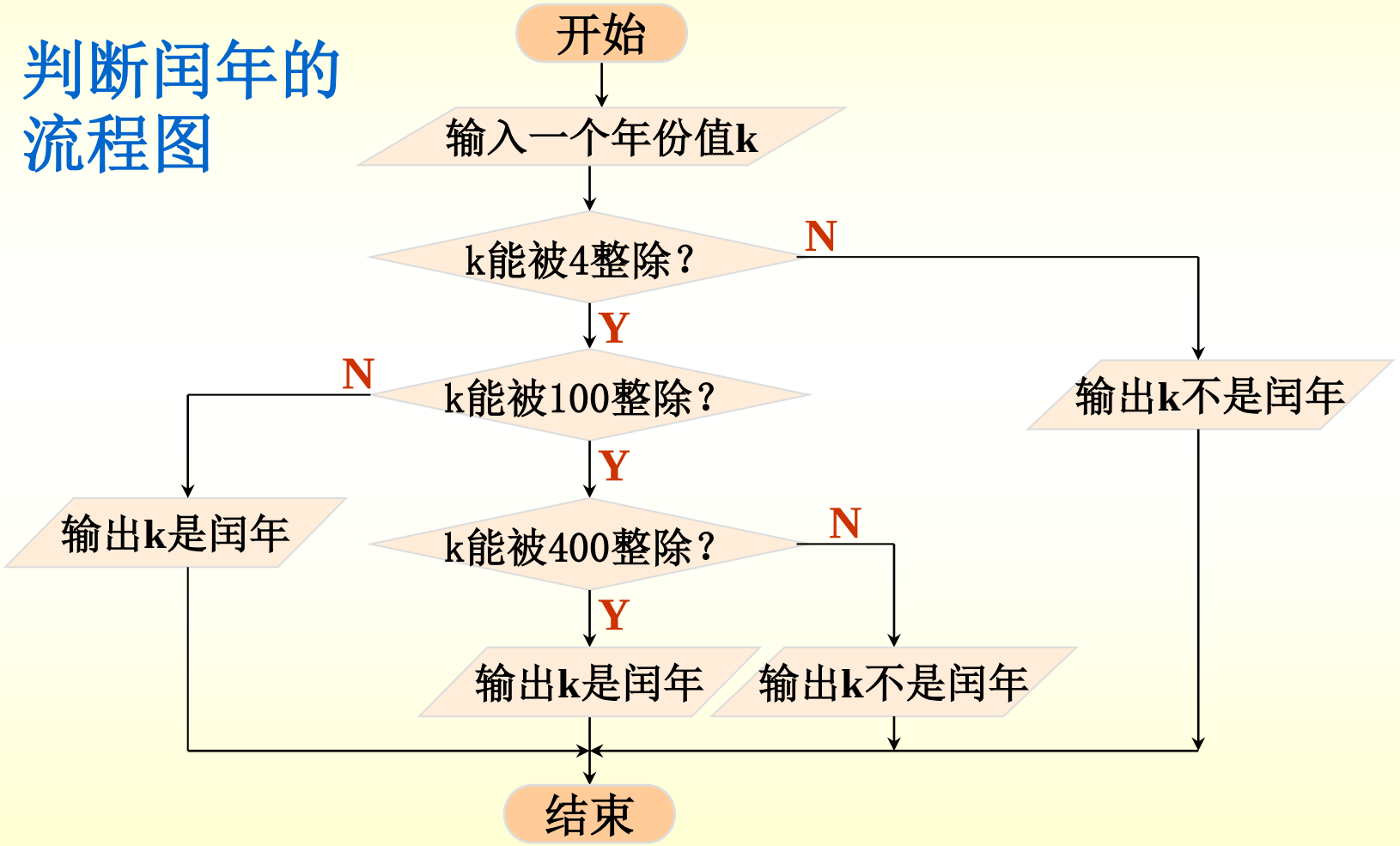
步骤3：若y能被100整除并且能被400整除，则转向步骤5；否则，转向步骤4；

步骤4：输出y不是闰年，算法结束；

步骤5：输出y是闰年，算法结束。

3、算法构造的过程

判断闰年的流程图



3、算法构造的过程

对于一个可以存在解决算法的问题，为了能够找到解决的算法思路，可以通过逐步求精（stepwise refinement）方法来进行。逐步求精就是通过把复杂问题不断分解为子问题，直到分解的子问题能够直接给出解决思路为止；然后再逐步将子问题的解决思路一层一层的整合起来，最终给出总问题的解决思路。问题不断分解的过程称为自上而下的分析方法，将思路不断整合的过程称为自下而上的综合方法。这也是编制复杂软件系统最为常用的两种策略，非常重要。

3、算法构造的过程

当然，逐步求精不是求解问题的全部，对于给定问题如何发现解决的算法，永远是一个开放的科学难题，需要人们不断地去探索和发展新的方法。

总之，算法发现是一项富有挑战性的技巧性工作，也是你们聪明才智得以展现的最佳场所。