第5章 计算思维与算法

计算思维是一种建立在计算机科学基础概念之上的思维活动。简而言之，计算思维要求我们能够像计算机科学家一样思考或解决问题。问题求解的过程包含以下步骤。

(1) 提出问题。针对实例问题进行综合归纳，并深入理解。

(2) 分析问题。抽象问题，建立模型，构造求解过程。

(3) 设计算法。设计详细的解题方法和步骤，并且利用计算机语言来实现求解过程。

(4) 解决问题。利用计算机来自动执行解决方案，测试调整，最终解决问题。

可见，与传统求解思维模式不同的是：要通过算法设计和编程来形成计算机可自动执行的命令。

5.1 算法的概述

5.1.1 算法的定义和由来

算法，就是指对问题解决方案进行准确和完整的描述，是解决问题的一系列清晰的指令。算法是一组完成任务的指令，能通过符合一定规范的输入，在有限时间内获得所要求的输出，任何代码片段都可视为算法。

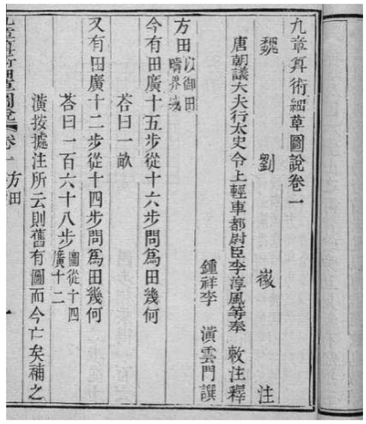
算法在中国古代文献中称为“术”，最早出现在《周髀算经》(图 5.1(a))和《九章算术》(图 5.1(b))中。特别是《九章算术》一书，给出了四则运算、最大公约数、最小公倍数、开平方根、开立方根、线性方程组等诸多算法。三国时代的刘徽也给出求圆周率的算法：刘徽割圆术。唐代的《算法》、宋代的《杨辉算法》、元代的《丁巨算法》、明代的《算法统宗》、清代的《开平算法》中也出现了算法论述。

算法的英文单词是 algorithm，最早由 9 世纪的波斯数学家 al-Khwarizmi(比阿勒·霍瓦里松)提出，随后传到了欧洲。在现代的数学认知中，欧几里得算法被西方人认为是史上第一个算法。

5.1.2 算法的特征

算法的五个重要特征如下。

(a)《周髀算经》



(b)《九章算术》

图 5.1 《周髀算经》和《九章算术》

1. 输入

一个算法必须有 0 个或以上输入量，所谓 0 个输入是指算法本身定出了初始条件。

2. 输出

一个算法应有一个或以上输出量，输出量是算法计算的结果，没有输出的算法是毫无意义的。

3. 明确性

算法的描述必须无歧义，算法里的每一步骤必须有确切的定义，不能含糊不清，不能有二义性，以保证算法的实际执行结果精确地符合要求或期望，通常要求实际运行结果是确定的。

4. 有限性

算法必须在有限个步骤内完成任务。

5. 有效性

又称可行性。能够实现，即算法中描述的操作都是可以通过已经实现的基本运算执行有限次来实现。

5.1.3 算法的描述

用来描述算法的方式主要有 5 种：自然语言、流程图、N-S 图、伪代码和程序设计语言。自然语言是人们日常使用的语言，可以是中文、英文等，用它来描述算法，人们不用进行专门的学习；流程图，是指用规定的图形符号来描述算法；N-S 图是一种完全去掉流程方向线的图形描述方法；伪代码即是用介于自然语言和程序设计语言之间的文字和符号来描述算法；程序设计语言是用计算机高级语言来描述求解过程，可以在计算机上运行。

举例：求解 1＋2＋…＋10000 的算法描述。

分析问题：求解 1＋2＋…＋10000 最直观的一种方法是顺序相加法，思路为：先求 1 加 2，得到结果 3，再加上 3，得到 6，以此类推，最终得到结果。

但顺序相加法并不是最优的算法，快速计算可以采用高斯算法中的首尾相加法，即

1＋10000＝10001；

2＋9999＝10001；

3＋9998＝10001；

……

相加 5000 次，得出最终答案。

1. 自然语言描述算法

自然语言描述算法，就是把算法的各个步骤，依次用人们熟悉的自然语言文字或符号表达出来。用自然语言描述算法的优点是通俗易懂，当算法中的操作步骤都是顺序执行时比较直观、容易理解。缺点是如果算法中包含了判断结构和循环结构，并且操作步骤较多时，就显得不那么直观清晰了。

顺序相加法的自然语言描述如下。

第 1 步：给变量赋初值，将 0 赋值给 S，1 赋值给 i；

第 2 步：S 的值加 i，得到 1＋2＋3…的效果；

第 3 步：每次循环需要让 i 自增 1，以达到从 1 加到 10000 的结果；

第 4 步：如果满足循环的条件，即 $i \leqslant 10000$，则转到第 2 步，否则转到第 5 步；

第 5 步：当 i 大于 10000 时，退出循环，输出 S 的值，算法结束。

首尾相加法的自然语言描述如下。

第 1 步：给变量赋初值，将 1＋10000 赋值给 S，10000/2 赋值给 i；

第 2 步：将 $S * i$ 的值赋值给 S；

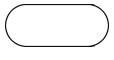
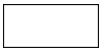
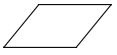
第 3 步：输出 S 的值，算法结束。

自然语言描述算法的特点是通俗易懂，简单易学，但不够简洁，易产生歧义。

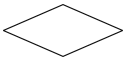
2. 传统流程图描述算法

传统流程图是用规定的一组图形符号、流程线和文字说明来描述算法，见表 5.1。用流程图描述算法就可以解决自然语言描述算法的缺点。结构化程序设计方法中规定的三种基本程序流程结构(顺序结构、选择结构和循环结构)都可以用流程图明晰地表达出来。例如，顺序相加法的传统流程图如图 5.2 所示。

表 5.1　流程图常用图形符号

图形符号	名　称	含　义
	起止框	程序的开始或结束
	处理框	数据的各种处理和运算操作
	输入/输出框	数据的输入和输出

续表

图形符号	名　称	含　义
	判断框	根据条件的不同,选择不同的操作
	连接点	转向流程图的他处或从他处转入
	流向线	程序的执行方向

3. N-S 流程图描述算法

虽然用流程图描述的算法条理清晰、通俗易懂,但是在描述大型复杂算法时,流程图的流向线较多,影响了阅读和理解。因此有两位美国学者 I.Nassi 和 B.Shneiderman 提出了一种完全去掉流程方向线的图形描述方法,称为 N-S 图(两人姓氏的首字母组合)。N-S 流程图完全去掉了流程线,用一个矩形框来表示一个算法,矩形框内可以包含若干从属于它的小矩形框,更清晰地表达结构化的算法设计思想,顺序相加法的 N-S 流程图如图 5.3 所示。

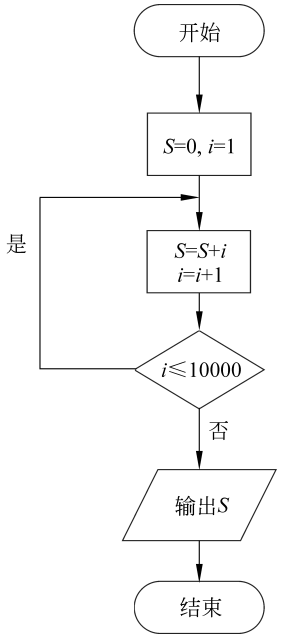


图 5.2　顺序相加法的传统流程图

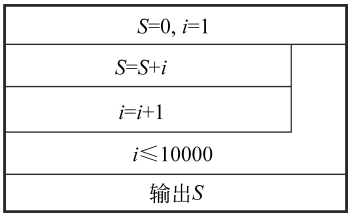


图 5.3　顺序相加法的 N-S 流程图

4. 伪代码描述算法

伪代码是用在更简洁的自然语言算法描述中,用程序设计语言的流程控制结构来表示处理步骤的执行流程和方式,用自然语言和各种符号来表示所进行的各种处理及涉及的数据。它是介于程序设计语言和自然语言之间的一种算法描述方法。这样描述的算法书写比较紧凑、自由,也比较好理解(尤其在表达选择结构和循环结构时),同时也更有利

于算法的编程实现(转化为程序)。伪代码没有固定、严格的语法规则,可以用英文,也可以用中文,更易转换为计算机高级语言。顺序相加法的伪代码描述如下。

```
S=0;
for(i=1; i≤10000; i++)
      S=S+i;
End
Output: S
```

注意:for 循环是编程语言中的一种循环语句,判断括号内条件是否成立,若成立,执行中间循环体,反之则退出此 for 循环。Output 为 S,即此程序执行完成后输出的结果为当前 S 的值。

5. 程序设计语言描述算法

算法最终都要通过程序设计语言描述出来(编程实现),并在计算机上执行。程序设计语言也是算法的最终描述形式。无论用何种方法描述算法,都是为了将其更方便地转换为计算机程序。由于本书下篇讲解 Python 编程,因此在此用 Python 语言来描述该顺序相加算法,程序代码如下。

```
S=0
for i in range(10001):
    S=S+i
print(S)
```

注意:for i in range 是 Python 中用来 for 循环遍历的,range()函数可创建一个整数数列,如 range(10)产生的数列为 0、1、2、3、4、5、6、7、8、9。print()函数用于打印输出结果。

5.1.4　算法的评价

1. 算法的评价标准

1)正确性

(1) 对输入数据能够得出满足要求的结果。

(2) 对一切合法输入,都可以得到符合要求的解。

2) 可读性

(1) 算法主要用于人们的阅读与交流,其次才是为计算机执行。

(2) 算法简单,则程序结构也会简单,便于程序调试。

3) 健壮性

算法应具有容错处理。输入非法数据或错误操作应给出提示,而不是中断程序执行;返回表示错误性质的值,以便程序进行处理。

4) 效率

(1) 每个问题有多个算法存在,每个算法的计算量都会不同。

(2) 在保证运算正确的前提下,力求算法简单。

2. 算法性能的度量

由于计算机的运算速度和空间资源有限,就需要花大力气去评估算法性能的好坏。衡量算法性能的好坏有多种度量标准,其中最重要的两大度量标准是时间复杂度和空间复杂度。

1) 时间复杂度

时间复杂度是描述算法运行时间的一个标准。时间复杂度常用大写字母 O 表示。O 的意思是"忽略重要项以外的内容",也就是将运行时间简化为一个数量级。时间复杂度的数量级表示有以下原则。

(1) 若运行时间为常数级,复杂度就记为 $O(1)$。

(2) 若不是常数级,就保留最高阶,同时省去最高阶的系数。如运行时间为 $T(n)=3n^3+4n^2+5$,则时间复杂度为 $O(n^3)$。

2) 空间复杂度

空间复杂度是描述算法空间成本的一个标准,即算法在运行过程中临时占用的存储空间大小的数量级。空间复杂度也使用大写字母 O 表示法。空间复杂度的数量级表示有以下原则。

(1) 若算法的存储空间大小固定,和输入规模没有关系,复杂度就记为 $O(1)$。

(2) 若算法分配的空间是一个线性集合(如列表),并且集合大小和输入规模 n 成正比时,复杂度记为 $O(n)$。

(3) 若算法分配的空间是一个二维列表集合(如二维列表),并且集合的长度、宽度和输入规模 n 成正比时,复杂度记为 $O(n^2)$。

算法的时间和空间复杂度是相互影响的。很多时候,需要在两者之间进行取舍。因此,设计一个算法时,要综合考虑算法的各项性能,如使用频率、处理数据量的大小、编程语言的特性、算法运行的机器系统环境等因素,才能够设计出比较好的算法。

5.2 常用经典算法

5.2.1 穷举算法

穷举算法又称列举法、枚举法、暴力破解法,是一种简单而直接的解决问题的方法。该算法简单,适应问题广,但计算量大。

穷举算法的步骤如下。

(1) 根据问题的具体情况确定穷举量(简单变量或数组)。

(2) 根据确定的范围设置穷举循环。

(3) 根据问题的具体要求确定约束条件。

(4) 设计穷举程序并运行、调试,对运行结果进行分析与讨论。

当问题涉及的数量非常大时,穷举的工作量也就很大,程序运行时间也就很长。因

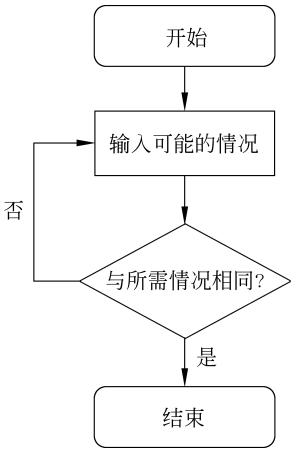


图 5.4　穷举算法流程图

此，应用穷举法求解时，应根据问题的具体情况分析归纳，寻找并简化规律，精简穷举循环，优化穷举策略。

穷举算法的流程如图 5.4 所示。

穷举算法的典型实例即百钱买百鸡问题，描述如下。

某人有 100 元，要买 100 只鸡。公鸡 5 元 1 只，母鸡 3 元 1 只，小鸡 1 元 3 只。问可买到公鸡、母鸡、小鸡各多少只。

解题思路：设公鸡 X 只，母鸡 Y 只，小鸡 Z 只，则有：

$$\begin{cases} X+Y+Z=100 \\ 5X+3Y+Z/3=100 \end{cases} \tag{5.1}$$

由此可以确定 X、Y、Z 的取值范围如下：

$$\begin{cases} 0 \leqslant X \leqslant 100/5 \\ 0 \leqslant Y \leqslant 100/3 \\ Z=100-X-Y \end{cases} \tag{5.2}$$

依次列举出可能的选项，如：

第一种：选择买 0 只公鸡，0 只母鸡，100 只小鸡。

第二种：选择买 0 只公鸡，1 只母鸡，99 只小鸡。

第三种：选择买 0 只公鸡，2 只母鸡，98 只小鸡。

……

百钱买百鸡的穷举法求解用 Python 语言编写如下：

```
for x in range(0, 21):                    #公鸡的上限 20
    for y in range(0,34):                 #母鸡的上限 34
      z =100 -x -y;                       #剩余小鸡
      if ((z %3 ==0) and (x * 5 +y * 3 +z / 3 ==100)):
          print("公鸡:{0}只,母鸡:{1}只,小鸡:{2}只".format(x, y, z));
```

程序运行后，输出结果如下：

```
>>> %Run bqmbj.py
  公鸡:0只, 母鸡:25只,小鸡:75只
  公鸡:4只, 母鸡:18只,小鸡:78只
  公鸡:8只, 母鸡:11只,小鸡:81只
  公鸡:12只, 母鸡:4只,小鸡:84只
```

5.2.2　贪心算法

贪心算法，又称贪婪算法，是指在对问题求解时，每一步都要选择当前看来最好的，做完此选择后，便将问题化为一个子问题。这是一个自顶向下的顺序求解过程，每一步都是单独考虑的。贪心算法并没有对所有可能解决问题的方法搜索，所以虽然它每一步都能保证获得局部最优解，但由此产生的最终解不一定是全局最优解。通常不能得到全局最

优解。贪心算法的步骤如下。

(1) 建立数学模型来描述问题。

(2) 把求解的问题分成若干子问题。

(3) 对每个子问题求解,得到子问题的局部最优解。

(4) 把子问题的局部最优解合成为原来问题的一个全局解。

贪心算法的流程如图 5.5 所示。

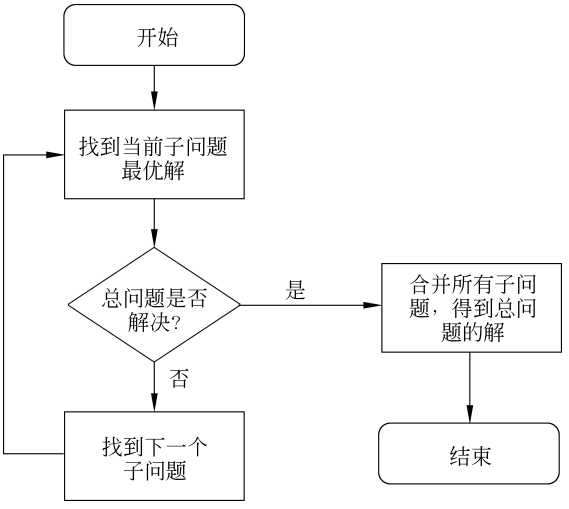


图 5.5 贪心算法流程图

贪心算法的典型实例即背包问题,描述如下。

有一个背包,背包容量是 $M=150$。表 5.2 所示有 7 个物品,物品可以分割成任意大小。要求尽可能让装入背包中的物品总价值 v 最大,但不能超过总容量 w。

表 5.2 背包物品重量和价值分配

物品	A	B	C	D	E	F	G
重量	35	30	60	50	40	10	25
价值	10	40	30	50	35	40	30

利用贪心算法来解决该问题,就是要在一定重量 w 内总价值 v 最高,每次选择物品都要选当前性价比高的,即按照性价比 v/w 的值从大到小来挑选,最后就会有最优解。贪心算法并不一定能得到全局最优解,而需根据实际情况灵活使用。

利用贪心算法求解该背包问题的 Python 程序如下:

```
#商品数据,使用元组列表,每个元组包含 (id, weight, value)
some_goods = [
    ('A', 35,10),
    ('B', 30, 40),
    ('C', 60, 30),
```

```
    ('D', 50, 50),
    ('E', 40, 35),
    ('F', 40, 35),
    ('G', 25, 30)
]

#背包容量
capacity =150

#按单位价值量排序商品
sorted_goods =sorted(some_goods, key=lambda x: x[2] / x[1], reverse=True)

#选择商品放入背包
selected_goods =[]
for id, weight, value in sorted_goods:
    if weight <=capacity:
        selected_goods.append((id, weight, value))
        capacity -=weight

# 打印结果
for id, weight, value in selected_goods:
    print('物品编号:{}, 放入重量:{}, 放入的价值:{}, 单位价值量为:{:.2f}'.format
(id, weight, value, value / weight))
```

程序运行后，输出结果如下：

```
>>> %Run goods2.py
  物品编号:B, 放入重量:30, 放入的价值:40, 单位价值量为:1.33
  物品编号:G, 放入重量:25, 放入的价值:30, 单位价值量为:1.20
  物品编号:D, 放入重量:50, 放入的价值:50, 单位价值量为:1.00
  物品编号:E, 放入重量:40, 放入的价值:35, 单位价值量为:0.88
```

5.2.3　递推算法

一个问题的求解需要一系列的计算，在已知条件和所求问题之间总存在某些相互关联的关系。计算时，如果可以找到前后过程之间的数量关系（即递推式），就能把复杂问题简单化，将其拆分成若干重复简单的运算，发挥计算机擅长重复处理的特点。

递推算法的首要问题是找出相邻的数据项间的关系（即递推关系）。递推算法避开了求通项公式的麻烦，把一个复杂问题的求解分解成了连续的若干简单运算。

递推算法的步骤如下。

（1）确定递推变量：应用递推算法解决问题，要根据问题的实际情况设置递推变量。

（2）建立递推关系：递推关系是指如何从变量的前一些值推出下一个值，或从变量

的后一些值推出其上一个值的公式(或关系)。递推关系是递推的依据,是解决递推问题的关键。

(3) 确定初始(边界)条件:对所确定的递推变量,要根据问题最简单情形的数据确定递推变量的初始(边界)值,这是递推的基础。

(4) 对递推过程进行控制:递推过程不能无休止地重复执行下去。递推过程在什么时候结束或满足什么条件结束,是编写递推算法必须考虑的问题。

递推算法的流程如图 5.6 所示。

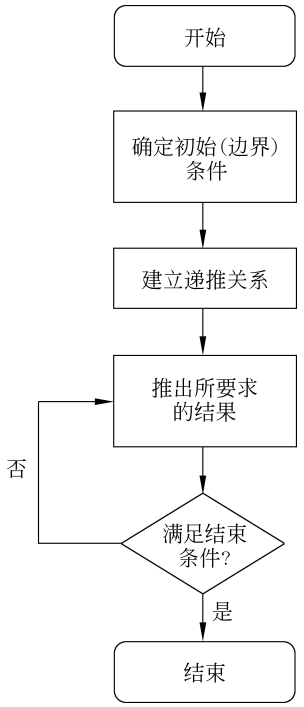


图 5.6 递推算法流程图

递推算法的典型实例即兔子繁殖问题,描述如下。

把雌雄各一的一对新兔子放入养殖场中。每只雌兔在出生两个月以后,每月产雌雄各一的一对新兔子。试问第 n 个月后养殖场中共有多少对兔子。

递推算法的关键是找出递推关系,每个月兔子的个数如下:

1 月:2

2 月:2

3 月:4

4 月:6

5 月:10

……

根据上述数据分析,可以得出每个月兔子的个数与上个月、上上个月的兔子个数的关系,即 $f(n)=f(n-1)+f(n-2)$,其中 $f(n)$表示第 n 个月的兔子数。这个式子就是递推关系式。

为了解决这个问题,可以按照以下步骤进行:

(1) 初始化兔子数量:假设初始兔子数量为 2 只(即第一对兔子为初始兔子)。

(2) 确定时间范围:确定繁殖的月份范围,即确定需要计算兔子数量的月份。

(3) 计算每个月的兔子数量:根据问题描述,每个月的兔子数量=上个月的兔子数量+上上个月的兔子数量,即 $f(n)=f(n-1)+f(n-2)$。

(4) 输出结果:输出每个月的兔子数量。

对应的 Python 程序编写如下:

```
f = [2]  #初始化兔子数量
months = 12  #确定时间范围

#计算每个月的兔子数量
for i in range(1, months):
    if i < 2:
        f.append(f[0])  #前三个月的兔子数量等于初始兔子数量
```