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。算法设计显得至关重要，它是计算思维的核心。

5.1 算法的概述

5.1.1 算法的定义和由来

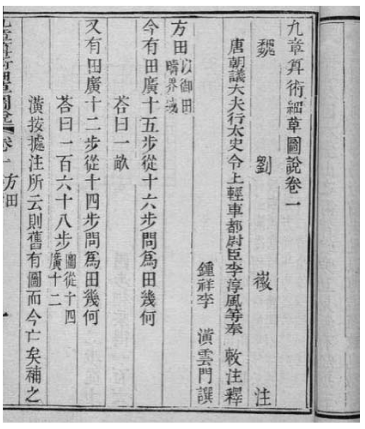
算法，就是指对问题解决方案进行准确和完整的描述，是解决问题的一系列清晰的指令。算法能通过符合一定规范的输入，在有限时间内获得所要求的输出。

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》


周髀算經序

趙爽 述

夫高而大者莫大於天厚而廣者莫廣於地體


九章算術細草圖說卷一

魏 劉 徽 注

唐朝議大夫行太史令上輕車都尉臣李淳風等奉 敕注釋

鍾祥李 潰雲門讎

方田 以御田疇界域

今有田廣十五步從十六步問爲田幾何

荅曰一畝

又有田廣十二步從十四步問爲田幾何

荅曰一百六十八步 圖從十四 廣十二

潰按據注所云則舊有圖而今亡矣補之

(a) 周髀算经　　(b) 九章算术

图 5.1 《周髀算经》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,国外实际最早由 9 世纪的波斯数学家 al-Khwarizmi 提出,随后传到了欧洲。在现代的数学认知当中,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。

1. 确定性

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。

2. 可行性

算法中执行的任何计算步骤都是可以被分解为基本的可执行的操作步骤,即每个计算步骤都可以在有限时间内完成。

3. 有穷性

算法必须能在执行有限个步骤之后终止。

4. 输入

算法有 0 个或多个输入,所谓 0 个输入,是指算法本身定出了初始条件。

5. 输出

算法必须有一个或多个输出,以反映对输入数据加工后的结果,没有输出的算法是毫无意义的。

5.1.3 算法的描述

用来描述算法的方式主要有以下 4 种:自然语言、流程图、伪代码和计算机语言。

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。流程图包括传统流程图和 N-S 流程图。伪代码即是用介于自然语言和计算机语言之间的文字和符号来描述算法。计算机语言即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，就是对算法的各个步骤，依次用人们熟悉的自然语言文字或符号表达出来。对于顺序相加法，算法如下。

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. 流程图描述算法

1）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，顺序相加法的传统流程图如图 5.2 所示。

2）N-S 流程图描述算法

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，可以更清晰地表达结构化的算法设计思想，如图 5.3 所示。

用 N-S 流程图表示算法直观易懂，但画起来比较费时。

思考：首尾相加法的流程图该如何画呢？

3）伪代码描述算法

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，虽然更易转换为计算机高级语言，但是不够直观。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。

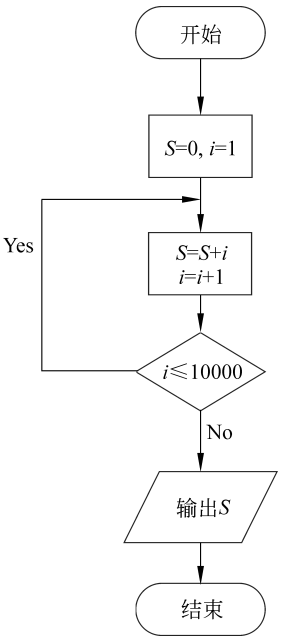


图 5.2　顺序相加法的传统流程图

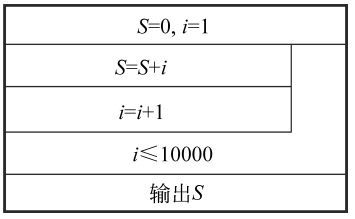


图 5.3　顺序相加法的 N-S 流程图

4）计算机语言描述算法

由于本书下篇讲解 Python 编程，因此在此只用 Python 语言来描述该顺序相加的算法。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

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的好坏。衡量算法的好坏有多种标准，其中最重要的两大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)，并且集合的长度和宽度和输入规模 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、处理的数据量的大小、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，计算量大，适应问题广。

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。

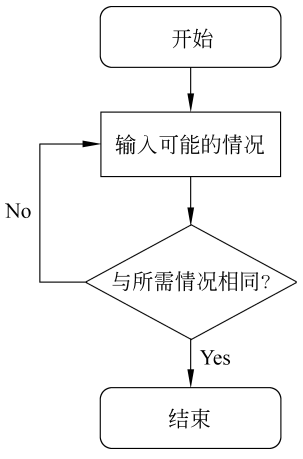


图 5.4 穷举算法流程图

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 元钱，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。

……

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。

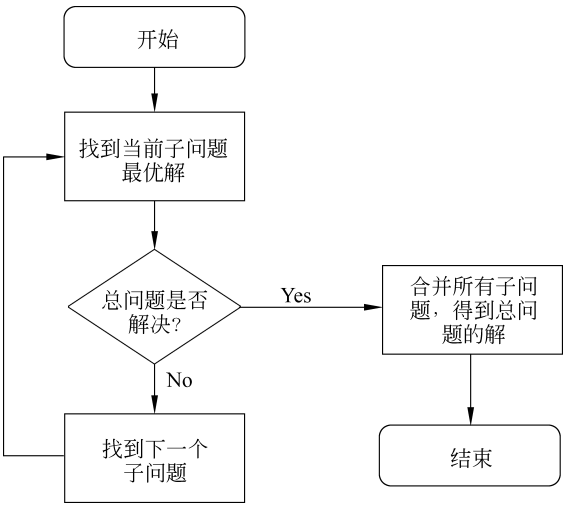


图 5.5 贪心算法流程图

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 5.1 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 5.1　背包物品重量和价值分配

物　品	重　量	价　值	物　品	重　量	价　值
A	35	10	E	40	35
B	30	40	F	10	40
C	60	30	G	25	30
D	50	50			

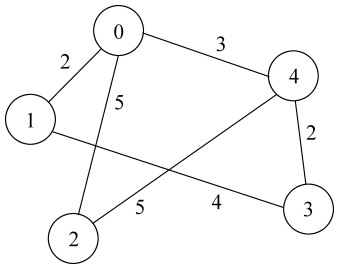


图 5.6　贪心算法示意图

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，需根据实际情况灵活使用。如图 5.6 所示，对从 0 点到 3 点的最短距离，按照贪心算法得出最短路径为 0→1→3，结果为 6，但我们可以轻易得出 0→4→3 的距离是最短的，结果为 5。

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 5.7 所示。

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。

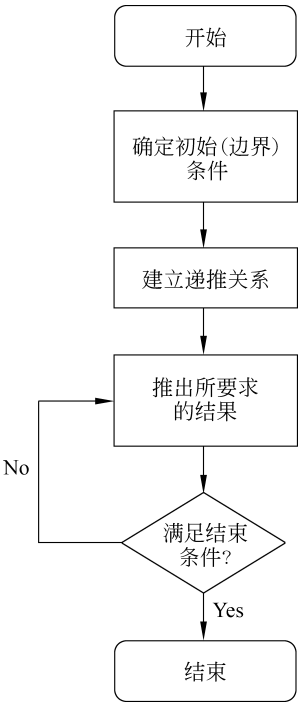


图 5.7　递推算法流程图

5.2.4　递归算法

递归就是子程序(或函数)直接调用自己或通过一系列调用语句间接调用自己的过程。

递归的两个基本要素如下。

(1) 边界条件：确定递归到何时终止，也称为递归出口。

(2) 递归模式：大问题是如何分解为小问题的，也称为递归体。

递归算法的步骤如下。

(1) 递推阶段：把较复杂问题(规模为 n)的求解推到比原问题简单一些的问题(规模小于 n)的求解。

(2) 回归阶段：当获得最简单情况的解后，逐级返回，依次得到稍复杂问题的解。

递归算法的流程如图 5.8 所示。

递归算法的典型实例即汉诺塔问题，描述如下。

有 n 个圆盘和 A、B、C 三根柱子，刚开始所有圆盘都在 A 柱子上，移动圆盘满足如下条件：①大圆盘不能放在小圆盘上；②一次只能移动一个圆盘，问至少移动几次才能将所有圆盘移动到 C 柱子上(可以借助 B 柱子)。

用递归方法来思考：将 n 个圆盘分为最后 1 个圆盘和上层 $n-1$ 个圆盘，先将上层 $n-1$个圆盘移到 B 上，再将最后 1 个圆盘移到 C 上，最后将上层 $n-1$ 个圆盘移到 C 即可。然后将上层 $n-1$ 个圆盘再分为上层 $n-2$ 个圆盘和最后 1 个圆盘，重复上述步骤……最后可将 n 层汉诺塔问题简化为两层汉诺塔问题，然后递归完成 n 层汉诺塔问题。

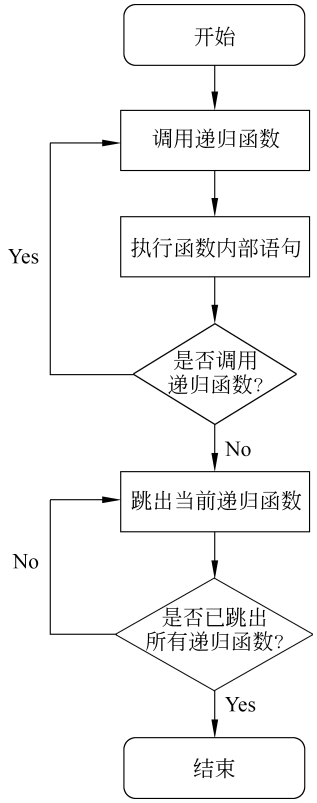


图 5.8　递归算法流程图

5.2.5　回溯算法

回溯算法类似于穷举法的思想，按照深度优先的顺序穷举所有可能性的算法。但是回溯算法比穷举法更高明的地方就是它可以随时判断当前状态是否符合问题的条件。一旦不符合条件，就退回到上一个状态，省去了继续往下探索的时间。满足回溯条件的某个状态的点称为“回溯点”。许多复杂的、规模较大的问题都可以使用回溯法，该方法有“通用解题方法”的美称。

回溯算法的步骤如下。

（1）针对所给问题，定义问题的解空间，它至少包含问题的一个（最优）解。

（2）确定易于搜索的解空间结构，使得能用回溯法方便地搜索整个解空间。

（3）以深度优先的方式搜索解空间，并且在搜索过程中用剪枝函数（注：若把搜索过程看成是对一棵树的遍历，那么剪枝就是将树中的一些“死胡同”和不能满足需要的枝条解“剪”掉，以减少搜索的时间。用约束函数在扩展结点处剪去不满足条件的子树，和用限界函数剪去得不到最优解的子树，这两类函数统称为剪枝函数。采用剪枝函数可避免无效搜索，提高回溯法的搜索效率）避免无效搜索。

回溯算法的流程如图 5.9 所示。

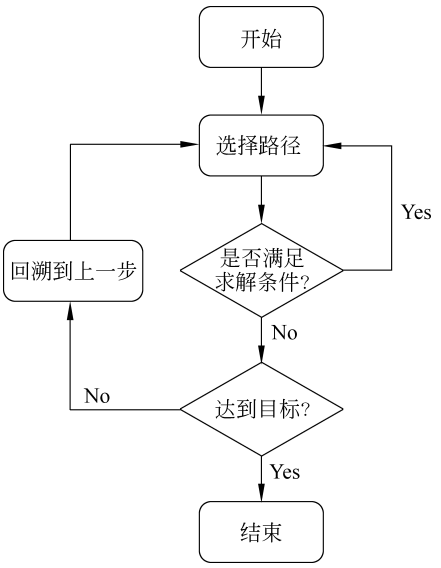


图 5.9　回溯算法流程图

回溯算法的典型实例即 n 皇后问题，描述如下。

$n \times n$ 格的棋盘上放置 n 个皇后，任何 2 个皇后不放在同一行、同一列或同一斜线上。问有多少种摆法。

用回溯算法来求解此问题：由于每行只能有一个皇后，第 1 行有 n 种可能，从第 1 行第 1 列开始，如果第 1 行第 1 列可以放置皇后，则找到下一行第 1 个能放置皇后的列。如果下一行没有符合条件的列，就返回上一行找到下一个可以放置皇后的列。遍历的行数等于 n，则获得一次结果。如果在第 1 行也找不到能放置皇后的列，则查找结束。使用回溯法时，当一条路可以前进时，就一直前进，行不通则退回上一步，以此类推。

5.2.6　动态规划算法

动态规划算法，要求问题具有最优子结构，是一种自底向上的求解思路。该算法将待求解的问题分解为若干子问题(阶段)，按顺序求解子阶段，前一子问题的解为后一子问题的求解提供了有用的信息。求解任一子问题时，列出各种可能的局部解，通过决策保留那些有可能达到最优的局部解，丢弃其他局部解。依次解决各子问题，最后一个子问题就是初始问题的解。

动态规划算法的步骤如下。

(1) 将问题按时空特性恰当地划分为若干阶段，定义最优指标函数，写出满足最优指标函数的表达式，以及边界/端点条件。

(2) 求解时从边界条件开始，逐段递推寻最优解。在每一个子问题求解时，都要使用它前面已求出的子问题的最优结果。最后一个子问题的最优解，就是整个问题的最优解。