第3章 计算思维

【学习内容】

本章介绍计算思维与计算机问题求解，主要知识点如下。

(1) 计算思维的定义及其核心概念。

(2) 逻辑思维与算法思维。

(3) 问题求解策略。

(4) 模式与归纳。

(5) 抽象与建模。

(6) 评价解决方案。

(7) 计算机问题求解的步骤。

(8) 算法、数据结构与程序设计语言的关系。

(9) 算法设计常用策略。

(10) 程序设计语言要素。

【学习目标】

通过本章的学习，读者应该掌握以下内容。

(1) 了解计算思维的定义与核心概念。

(2) 理解分解法问题求解策略。

(3) 理解模式与归纳方法。

(4) 理解科学抽象方法与原则。

(5) 理解计算机问题求解的一般步骤。

(6) 理解算法的概念、算法的描述方法。

(7) 了解常用的算法设计策略和常用典型算法。

(8) 了解常用的数据结构及其特点。

(9) 了解算法复杂度等评价标准。

(10) 了解软件开发方法及其应用。

(11) 了解典型算法的 Python 实现。

计算思维应成为信息社会每个人必须具备的基本技能。本节将围绕计算思维的核心概念——逻辑思维、算法思维、问题求解策略、模式与归纳、抽象与建模、解的评价，以及算法、数据结构与程序等内容展开，为读者利用计算思维解决各领域问题奠定基础。

3.1 概　　述

1.5节中简要介绍了计算思维及其重要性，但目前仍没有一个明确而统一的计算思维的定义，不同的学者从不同的角度来定义计算思维，这些定义有很多相似的地方，但也有差别。例如，周以真认为，计算思维是定义和解决问题，并将其解决方案表达为人或机器能有效执行的形式的思维过程。也有学者认为计算思维是对问题进行抽象并形成自动化解决方案的思维活动，等等。因此，理解并学习计算思维的最好可行方法，是从各类定义中提取出共有的核心概念，识别出那些外围的、不应包括在计算思维定义中的非核心概念，然后聚焦于核心概念的学习。

通常认为计算思维的核心概念有以下一些，本章将在后面具体介绍。

(1) 逻辑思维。

(2) 算法思维。

(3) 分解。

(4) 泛化与模式识别。

(5) 建模。

(6) 抽象。

(7) 评估。

而以下一些概念一般不认为是计算思维的核心概念。

(1) 数据表示。

(2) 批判性思维。

(3) 计算机科学。

(4) 自动化。

(5) 模拟与可视化。

计算思维不是计算机科学家所特有的，而应该成为信息社会每个人必须具备的基本技能。计算思维已经在其他学科中产生影响，而且这种影响在不断拓展和深入。计算机科学与生物、物理、化学，甚至经济学相结合，产生了新的交叉学科，改变了人们认识世界的方法。例如，计算生物学正在改变生物学家的思考方式，计算博弈理论正在改变经济学家的思考方式，纳米计算正在改变化学家的思考方式，量子计算正在改变物理学家的思考方式。

任何人都可以应用计算思维，利用计算机来解决其领域(不止是计算机科学)的问题。计算思维的核心概念对不同领域的人来说是不同的。例如，对计算机科学家来说，算法思维指的是对算法及其在不同问题上的应用进行研究；对数学家来说，算法思维可能意味着按照运算规则进行某一种运算；对科学家来说，算法思维可能指的是按照某个流程进行一次实验。再例如，用隐喻和比喻写故事，对语言学家来说就是使用了抽象，当科学家构建了一个模型，或数学家使用代数描述问题时，就意味着使用了抽象。

考虑这样一个场景，在各类表彰大会上，一般都要给被表彰者颁发奖状或证书，由于

主席台场地所限，一次只能上去一部分人。每组获奖者在领取证书后，要拍照，然后退场。为使颁奖过程尽量高效，通常在一组人上台领奖时，下一组就已经在台下等候了。上一组人一离场，下一组人直接从台下登台重复领奖过程。这就是计算思维中流水线的应用。

也可从另一个角度来理解计算思维——计算思维不是什么，通过对这个问题的讨论和研究，能深入地理解计算思维。

首先，学习计算思维不等于学习计算机科学。后者学习的主要目标是学习和运用数学计算原理，来研究与计算机科学自身紧密相关的问题。前者的学习目标不是“像计算机科学家一样思考”，而是希望能用计算思维的核心概念，帮助解决日常生活或工作中的问题。其次，学习计算思维不是学习程序设计，后者的主要目的是如何更好地写程序，产出高质量的软件。前者会包含一部分程序设计的内容，但最主要的目标是如何在使用计算机的条件下，进行问题求解。

因此，学习计算思维的最终目标不是让每个人都像计算机科学家一样思考，而是能应用计算思维的概念，在所有学科领域内或学科之间解决问题、发现新问题。

此处，用一个简单的例子来展示利用计算思维求解问题与常用数学思维的区别。

例 3-1 有一些数，除以 10 余 7，除以 7 余 4，除以 4 余 1，求满足条件的最小正整数。

解：这些数满足的条件是比 10 的倍数小 3，且比 7 的倍数小 3，且比 4 的倍数小 3。即比 10、7 和 4 的公倍数小 3 的数都会满足条件。因此，最小的满足条件的数为 10、7 和 4 最小公倍数减 3，即 140－3＝137。

如果利用计算思维解这道题，核心是设计能自动执行的算法。思路是从 1 开始，不断地枚举数并判断枚举的数是否满足题设条件。不满足条件，则增 1 枚举下一个数。重复该过程，直到找到第一个满足条件的数，且第一个满足题目要求的数就是最小的数。按照该思路编写的 Python 程序代码如下所示。这个程序利用题设条件做了优化，即该数除以 10 余 7，则这个数的个位数是 7。因此，从 7 开始，每次增加 10，获得下一个数。该程序的输出为 137，与求最小公倍数的方法得到的结果一致。

```
for i in range(7, 1000, 10):
    if i%7 ==4 and i%4 ==1:
        print(i)
        break
```

在展开介绍计算思维的核心概念之前，先用一个简单的问题来说明各核心概念在问题求解时的体现。条形码是日常生活中常见的符号，如出现在各种商品包装上的条形码，图 3-1 是一个条形码示例。条形码的数字不是随便生成的，有一个公式，对条形码各位数字进行运算，如果条形码正确，则得到的结果是 10 的倍数。当条形码有错误（扫码器读入有误或印刷有错）或相邻两位数字调换了位置，则组合后的结果将不是 10 的倍数，这样就检测出了条形码中的错误。

图 3-1 一个条形码示例

具体的计算过程：从条形码最右边的数字开始，逐个取出数字，轮流用 1 和 3 乘以取

出的数字，然后把所有的乘积累加，看看得到的结果是否是 10 的倍数。对图 3-1 的示例，最右边数字是 9，则计算 9×1，9 的下一位数字是 4，则计算 4×3。照此依次计算 5×1、4×3、6×1、7×3、8×1 和 9×3，累加这些乘积，即 9＋12＋5＋12＋6＋21＋8＋27＝100，为 10 的倍数，因此，可以判断这个条形码是正确的。

假设扫码器将图 3-1 最左边的数字扫成了 5，则最后的计算结果是 9＋12＋5＋12＋6＋21＋8＋15＝88，不是 10 的倍数，则检测到扫码错。又例如，如果图 3-1 最左边两位数字印刷反了，变成了 8976 4549，则最后的计算结果是 98，也会检测到条形码的错误。

下面的代码给出了检测条形码正确性的 Python 实现。

```
barcode = input("请输入 8 位条形码: ")
total = 0
multiplier = 1
position_in_code = len(barcode)
while position_in_code != 0:
    total += int(barcode[position_in_code-1]) * multiplier
    if multiplier == 1:
        multiplier = 3
    else:
        multiplier = 1
    position_in_code -= 1
if total % 10 == 0:
    print("条形码扫码正确!")
else:
    print("条形码有错误!")
```

程序中将条形码以字符串的形式读入，然后 position_in_code 从最右边开始扫描，每扫描一个字符，根据当前的状态决定是乘 1 还是乘 3，并将字符转换成整数后再进行乘运算。

这个程序从设计到实现，处处都体现了计算思维的核心概念，总结于表 3-1 中。

表 3-1　计算思维的核心概念在条形码例子中的体现

核心概念	定　义	本例中的体现
逻辑思维	逻辑是研究推理的科学，逻辑思维帮助人们理解事物、建立和检查事实	逻辑让人们能读懂程序，理解其检测错误的工作原理。推理能用于预测检查正确或不正确的条形码时会发生什么
算法与算法思维	算法是用于完成某个任务的一系列指令或一组规则。算法思维能让人们设计出借助计算机解决问题的算法	能帮助人们理解利用公式验证条形码正确性的指令序列。也可以设计其他的算法完成同样的验证
分解	分解是将问题或系统拆分成更小、更易管理的小问题或小系统的过程。它有助于人们解决复杂的问题，因为拆分出的每个小部分都可以被单独解决，组合它们的解决方案后，可以构造出整个问题的解，而不需要一次考虑整个问题的解	把“在条码中查找错误”这个问题分解成多个小问题，例如，把条码解码成数字，把这些数字相乘和相加，检查运算结果，以及如果有错误或问题时给出错误信息

续表

核心概念	定　义	本例中的体现
泛化与模式	使用模式意味着发现解决方案之间的相似之处和共同的差异。通过提取模式，可以做出预测、制定规则并解决更一般的问题，这就是泛化	一个算法就能发现条形码中的典型错误，算法能够发现交替乘 1 和 3 的操作模式，用循环来消除重复的交替乘操作
抽象	抽象是通过识别和丢弃不必要的细节来简化事物，它使人们能处理复杂的问题。数据可以通过抽象来表示（例如，用数字来表示文本、图像、声音等）。模拟提供了真实场景的抽象	不同粗细的条纹表示不同的数字，一串数字表示一种产品。此外，该程序用交替乘 1 和 3 来构造校验和，而其他场合使用不同的校验和（例如，信用卡号码的校验和是通过乘以 2 来构造的，而 ISBN-10 号码用每个数字乘以不同的数来构造校验和），所以，可以抽象出使用不同数字构造校验和的算法
评估	评估指以客观的和系统的方式做出判断，并评估问题的可能解决方案。其目标是利用现有资源实现最有效率和最有效的解决方案	对该算法可以考虑这些评估问题：是否会发现所有可能的条形码错误？用该程序发现错误要花多长时间？一台小型机器能否快速高效地执行这个检错操作？

3.2　逻辑思维与算法思维

逻辑思维和算法思维都是计算思维的核心概念，处于非常重要的地位。关于逻辑和算法的内容，可以写成很厚的书籍。本节不会完备地介绍逻辑与算法相关的知识，而会关注于如何使读者形成逻辑思维和算法思维习惯的一些核心要素。

3.2.1　逻辑思维

简单来说，逻辑是关于推理的科学，是一种区分正确和不正确论证的系统。所谓论证，指的是从假设出发，经过一系列推理，得出结论的过程。逻辑包含一组规则，当将这组规则应用于论证时，能证明什么是成立的。下面是一个典型的逻辑论证证明。

1. 苏格拉底是人。
2. 所有的人都会死。
3. 所以，苏格拉底会死。

即便是没学过哲学和逻辑，也能看懂这个论证证明。这样的推理过程在日常生活中经常被使用，例如，房间里有风，房间的窗户是开着的，所以，风是从窗户吹进来的。但是，这样的推理并不能保证总是对的，有时会导致错误的结论。同时，计算思维需要利用计算机自动地完成各种推理，因此，在学习如何得出计算的解之前，要学会如何正确地运用

逻辑。

在逻辑推理中，所有已知的事实称为前提。前提的表示形式是带有真假含义的陈述句，因此，每个前提都对应一个值——“真”或“假”。在前面的苏格拉底论证中，前两句话就是前提——对应着“真”的两个陈述句。

有了前提后，就可以在此基础上进行推理，得出结论。推理的方法通常分为演绎推理和归纳推理。演绎推理从前提条件开始，推导出其结论，是最强的推理形式。初中数学中的几何证明，就是典型的演绎推理，从已知条件、公理和定理出发，证明题目结论的成立。例如，三角形内角和为 180°，$\triangle ABC$ 是一个三角形，所以，$\triangle ABC$ 内角和为 180°。

但是，也要注意在使用演绎推理时常犯的两种错误。第一种是前提有错，导致结论错误。例如下面这个推理，第 2 个前提是错误的，所以，推理过程即便是正确的，但是结论是错误的。

1. 卡拉是条狗。
2. 所有的狗都是棕色的。
3. 所以，卡拉是棕色的。

第二种错误是因为结论与前提没有必然联系导致的，例如下面这个推理，其过程是错的，所以导致结论是错的。

1. 所有的乒乓球是圆的。
2. 地球是圆的。
3. 所以，地球是乒乓球。

现实生活中其实很少使用演绎推理，因为演绎推理对所使用的前提要求较高，最好是简洁、清晰的。但是，现实生活中碰到的情况主要是混杂而不明晰的，所以，常用的推理是归纳推理。例如，直角三角形的内角和是 180°；锐角三角形的内角和是 180°；钝角三角形的内角和是 180°；直角三角形、锐角三角形和钝角三角形是全部的三角形；所以，一切三角形的内角和都是 180°。这个例子从直角三角形、锐角三角形和钝角三角形内角和分别都是 180°这些个别性知识，推出了“一切三角形内角和都是 180°”这样的一般性结论，这就是归纳推理。又例如，在数学题目中看到数列 0、2、4、6、8、10…时，会很自然地归纳出这个数列是由所有非负偶数构成的，据此，有很大把握归纳出 10 后面的数应是 12。

归纳推理对前提的要求较低，不一定是绝对的“真”或绝对的“假”，可以是“百分之多少的真”。因此，得到的结论也不能保证是绝对的“真”或“假”，而是有一定可能可信的结论。例如，下面的推理是正确的，在结合了概率知识后，结论是可信的。

1. 一个袋子里有 99 个红球和 1 个黑球。
2. 有 100 个人从袋子里取球，每人只能取 1 个。
3. 小明是 100 人中的 1 人。
4. 所以，小明有很大可能取到一颗红球。

逻辑推理对掌握计算思维非常重要，因为最后的解决方案要在计算机上运行，计算机给出的结果严格依赖于解决方案中的逻辑推理。因此，应用计算思维设计问题的解时，必

须保证以下 3 方面。

(1) 推理是有效的。

(2) 给计算机的输入是可靠的。

(3) 能解释计算机的输出,是绝对正确的(演绎推理)还是可能正确的(归纳推理)。

日常生活中大量使用的是归纳推理,得出的结论通常是“百分之多少的真”,而计算机更擅长处理非黑即白的绝对性问题。因此,为指示计算机进行逻辑决策,需要一种逻辑系统,能将日常的推理系统映射到计算机能理解的形式上。布尔逻辑即是这样一种逻辑系统,其处理的逻辑只能在“真”和“假”中取一个值。“真”和“假”在不同的问题背景下有不同的形式,例如,Python 中的 True 和 False,数字电路里的 0 和 1,等等。

布尔逻辑中的语句称为命题,命题指的是具有明确真假含义的陈述语句,其含义只能为真(True)或假(False),不能同时既为真又为假。其含义必须是明确的,而不是含糊的。可以将这种陈述句进行符号化,即用一个字母来表示一个命题,这样的字母称为命题词。多个命题结合起来构成更复杂的命题,称为复合命题,而连接命题的符号称为逻辑运算符。

下面以五子棋为例,介绍布尔逻辑的相关知识。五子棋游戏是大家基本玩过的棋类游戏(见图 3-2),它是一种两人对弈的纯策略型棋类游戏,棋具与围棋通用,是起源于中国古代的传统黑白棋种之一。五子棋专用棋盘为 15×15,盘面由纵横各十五条等距离垂直交叉的平行线构成,共 225 个交叉点,交叉点即为落子点,盘面正中一点为“天元”。

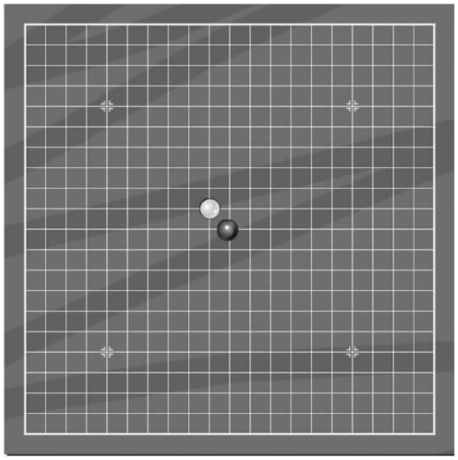


图 3-2　五子棋盘

五子棋的行棋顺序为黑先白后,从天元开始相互顺序落子。某些规划中,对双方前两步的行棋有约束,可分解为以下几条规则,可见每条规则是一个命题。每个命题后面的字母为其对应的命题词,注意字母的选取是任意的,也可用其他字母来表示这些命题。

1. 黑方的第一个棋子应下在天元上。(P)
2. 白方第一个棋子只能下在与天元邻近的 8 个点上。(Q)
3. 黑方第二个棋子只能下在与天元邻近的 5×5 点上。(R)
4. 白棋第二个棋子不受限制,可下在棋盘任意位置。(U)

可知这些条件必须同时成立，才是合规的下棋过程，即应该表达为“黑方的第一个棋子应下在天元上；并且白方的第一个棋子只能下在与天元邻近的 8 个点上；并且黑方第二个棋子只能下在与天元邻近的 5×5 点上；并且白棋第二个棋子不受限制，可下在棋盘任意位置”。用来连接这种所有条件必须同时成立的“并且”，是一种逻辑运算符，称为“与”(And)，对应着 Python 中的 and 运算符。上述行棋规则可用 Python 表达式表达为 P and Q and R and U。

五子棋判断胜负的规则较多，包括以下几条，每个命题后面的字母为其对应的命题词。

1. 最先在棋盘横向、竖向、斜向形成连续的相同色 5 个棋子的一方为胜。(P)
2. 黑棋禁手判负，白方应立即向黑方指出禁手，自然而胜。(Q)
3. 当黑方在落下关键的第五子时，若在形成五连的同时，又形成禁手，则禁手失效，黑方胜。(R)
4. 黑方走出长连禁手时，白方无论何时指出，立即获胜。(U)

可知这些条件只要有一条成立，则棋局以某一方获胜而结束。因此，上述规则又可表达为“最先在棋盘横向、竖向、斜向形成连续的相同色 5 个棋子的一方为胜；或黑棋禁手判负，白方应立即向黑方指出禁手，自然而胜；或当黑方在落下关键的第五子时，若在形成五连的同时，又形成禁手，则禁手失效，黑方胜；或黑方走出长连禁手时，白方无论何时指出，立即获胜”。用来连接这种所有条件至少有一条成立的“或”，称为“或”(Or)逻辑运算符，对应着 Python 中的 or 运算符。上述行棋规则可用 Python 表达式表达为 P or Q or R or U。

除了上述五子棋规则外，其实还隐含着一些显而易见、一般不做专门说明的规则。例如，如果棋盘上某个交叉点没有棋子，则该交叉点称为空闲交叉点。在遵循行棋规则的前提下，只能将子落在空闲交叉点上。此处，“没有棋子”就是对“有棋子”的否定，此处“没”是一种逻辑运算符，称为“非”(Not)，对应着 Python 中的 not 运算符。假设“有棋子”被符号化为 P，则“没有棋子”可用 Python 表达式表达为 not P。

上述用字母表示命题，用运算符表示自然语言中的连词的方法，称为符号化，它是符号逻辑的基础。符号逻辑是用数学方法研究逻辑或形式逻辑的学科。

3.2.2 算法思维

逻辑系统给出了一组规则用来对(一部分)现实世界进行建模和推理，既可以描述静态的事物，也可以描述事物的发展变化过程。例如，3.2.1 节的五子棋游戏，用布尔逻辑描述了行棋规则和输赢判断规则，通过将这些逻辑规则与具体棋局结合，可以使逻辑命题在具体棋局中做出真假判断，从而得出棋局在不同时刻的状态——谁赢了，是否还可继续落子，等等。

算法构建于逻辑之上，但不等同于逻辑，算法既要基于逻辑做出逻辑判断，又要基于逻辑判断执行某些动作，算法是现实世界计算系统的根本。

计算机执行算法时，只会关注当前正在执行的步骤，而在进入下一步时，会忘记上一

步做的所有事情。因此，要让算法正确执行，必须提供一种机制记住上一步或前几步做的事情。算法在执行过程中有其需要处理的对象，如参与加运算的加数。因此，本质上，只要记住每一步完成后，算法操作对象的值，即记住了该步所做的操作。这种用于保存操作对象结果的机制称为变量，变量的值可通过赋值进行修改。这种每一步都需要记住的信息称为状态，它是算法执行过程中需要记住的所有事物的总和。如五子棋游戏中，当前棋盘黑子和白子的位置、轮到谁落子等信息，就是五子棋算法需要记住的状态信息。

第 1 章已经介绍过，算法中动作步骤的组织可有 3 种方式：顺序的、选择的、循环的。顺序结构是最简单的、最常用的，只要按照解决问题的顺序写出相应的动作即可，动作执行顺序是自上而下，依次执行。

日常生活中有很多做事情的动作就是典型的顺序结构，例如，吃小笼汤包的十二字口诀“轻轻提、慢慢移、先开窗、后吸汤”，按照这个顺序，就能完成吃一个小笼汤包的动作。非常重要的一点是顺序非常重要，这 4 个动作的顺序一旦发生变化，吃小笼汤包的事就不会太顺利。

在地铁站自动售票机上买车票的过程，也是典型的顺序结构，如下所示。

1. 选择目的地。
2. 选择车票张数。
3. 投入硬币或插入合适的纸币。
4. 取找回的零钱。
5. 取车票。

现实世界中，还有许多场合需要先做逻辑判断，再根据判断，在多个动作中选择一个执行，此时就需要用到选择结构。例如，学校鼓励学生帮家里做家务，考虑这样一个场景，家长让你到楼下的小超市买一瓶酱油：如果有李锦记的生抽，就买李锦记的，没有就买其他牌子的也行。这段话是指导你购买酱油的算法，重新编排一下，用选择结构进行组织，如下所示。进入超市后，购买何种酱油是基于“是否有李锦记生抽”的判断结果的，如果有，则买李锦记的；如果没有，则买其他品牌的。但是不论如何，只会买一种酱油，而不会两种都买。

1. 如果超市有李锦记生抽
   买李锦记生抽。
2. 否则
   买其他品牌生抽。
3. 带上酱油，离开超市回家。

在使用选择结构组织动作序列时，必须考虑全部可能的情况。特别是当算法要由计算机来执行时，一定要记住计算机只会按你规定的动作执行，而不会帮你补充你遗忘的动作，也不会对你给的含糊描述的动作做出额外的解释。

假设家长让你到楼下小超市买酱油，只吩咐了买李锦记生抽。那么，如果当你到超市时，发现没有李锦记生抽，你会怎么办？打电话回去问，还是自己随便买一瓶？假设让计算机去买酱油的话，如果没有李锦记生抽，计算机将留在超市，再也回不来了。原因就是在设计买酱油的算法时，只考虑有李锦记生抽的情况，而没有告诉计算机，如果没有李锦

记生抽怎么办。计算机只能执行你告诉它的动作，因此，如果超市没有李锦记生抽，它就再也回不来了。

在日常生活中，很多情况下需要重复做同一个动作，只到达到了某个条件，才会停下来。例如，学过的急救知识中，对呼吸、心跳不规则或停止的病人，需马上进行体外心脏按压和口对口的人工呼吸。体外心脏按压和人工呼吸这个动作会重复做下去，直到病人心跳恢复，或救护车到来。

对这些需要重复做多次的动作序列，可用循环结构来组织。通常用一些条件来控制重复执行的动作序列，当达到停止条件时，可以结束重复动作的执行。这些停止条件即为循环控制条件。通常有两种方式来决定重复做的事是否能停下来：一种是计数器控制；另一种是标志值控制。

通常，为了加深对知识点的理解，老师要求学生多遍抄写某些重点知识点，假设是 10 遍。那么，抄写的过程可以用下面的动作序列来描述。其中，“抄写次数”即为计数器，当它达到 10 时，就不要抄了。注意，也可以一开始就把抄写次数设成 10，每抄写一遍，次数减 1。那么，此时的循环结束条件是什么呢？下面是抄写知识点的算法描述。

1. 抄写次数是 0。
2. 当抄写次数没到 10，重复做下面的动作。
   ① 抄写一遍知识点。
   ② 抄写次数加 1。
3. 完成作业。

下五子棋时，双方轮流落子，这个动作会重复执行下去。没办法像抄书一样，预先就知道下多少个回合就会结束棋局。因此，很难设一个计数器来控制双方落子次数。此时，可以通过设置一个标志值来控制落子的动作，这个值可以设成“黑胜或白胜”，黑胜或白胜的判断可用 3.2.1 节的规则判断，此处不再展开。下棋的过程即可用下面的算法来描述。

1. 当不是黑胜或白胜时，重复做下面的动作。
   ① 黑落子。
   ② 白落子。
2. 宣布获胜方。

算法的有限性要求算法在有限步内必须停下来，那么，当算法中出现了循环时，必须仔细设置循环的结束条件，确保循环总能停下来。通常称没法停下来的循环为无限循环，算法中的无限循环通常是一种错误(Bug)。

3.2.3 小结

逻辑思维和算法思维对掌握计算思维非常重要。在设计问题解决方案时，必须对逻辑有很好的理解，并能将自然语言描述的命题等正确地转换成符号逻辑。逻辑同时是算法的基础，也是计算的基础。算法描述的是过程性知识，这是利用计算思维设计问题解决方案的基石。因此，需要掌握算法思维来正确地组织解决方案的动作序列。

3.3 问题求解策略

问题求解是一种创造性的工作，有时确实需要“灵光一闪”，好像得到解的过程无迹可寻。但是，前人总结出来的系统化的求解问题的策略，对这种创造性的活动有很大的帮助。此外，需要借助计算机求解的问题，通常不是人们用纸和笔就能解决的小规模问题，因此，如何寻找大规模问题的解，分解策略就显得非常重要，也就不难理解其在计算思维中的核心地位了。

3.3.1 基本步骤

人进行问题求解的过程可归纳为以下的步骤[①]。

(1) 理解问题：输入是什么，输出是什么。

(2) 制订计划：准备如何解决问题。

(3) 执行计划：具体解决问题。

(4) 回头看：检查结果……

对上述问题求解的步骤逐条进行考察，看看计算机能在每一步做些什么事：

(1) 理解问题：计算机如何理解问题？

(2) 制定计划：计算机能制订计划吗？如果不能，如何针对计算机制订计划？即什么样的计划可能在计算机上实现？什么样的形式才能让计算机知道该做什么和怎么做？

(3) 执行计划：只有这个才真正是计算机能做的。

(4) 回头看：为什么结果是(不)正确的？求解效率还能提高吗？

因此，在理解问题并设计问题解时，解必须适合于在计算机上运行，并能发挥计算机的威力，这种解通常以算法形式给出。此外，还可以看出，计算机只能完成人给它规定的动作，而不要指望它做额外的事情。这也是运用计算思维求解问题时，需要特别注意的。

进行问题求解时，首先，不要害怕问题的规模和复杂度，复杂的大问题通常能被分解成相关的小问题，而这些小问题是较易解决的。其次，不要一拿到问题就开始写解决方案，这样做会导致对问题认识不清，可能会在解题过程中走偏，在错误的方向上越走越远。因此，不论何种情况，都必须先理解问题。

问题求解就是要在问题(起点)和答案(终点)之间建立一个连接。理解问题阶段就是要认清问题的起点和终点是什么，对起点的认识越深入，越能帮助人们发现什么是终点，以及如何到达终点。所谓终点，指的是要达成的目标，而不是如何达成。在理解问题阶段，要避免过早陷入寻找“如何做”中。

理解问题需要用到多方面的知识，如阅读理解、问题相关的背景知识、数学知识、物理知识，等等，这些需要经过学习不断地积累，非一日之功。有些用于认识问题的建议，可以

① 源自 George Polya 的 *How to Solve It* 一书

帮助我们有效认识问题。

(1) 对拿到的问题,尽量用自己的话再陈述一遍。

(2) 尽量用图表或其他图形方式重新描述问题。

(3) 尽量明确哪些是已知的,哪些是未知的,把未知的尽量明确。

(4) 给出问题解决的标准,即明确“如何知道问题被解决了”,这常常就是问题求解的目标。

(5) 尽量用可度量的词汇来描述问题求解目标。

(6) 大略地描述一下有效的解决方案看起来是什么样的,需要具备什么特征,等等。

对问题有很清楚的认识之后,可以开始制订解题计划。在制订解题计划过程中,有一些需要时刻牢记的原则。

(1) 解的质量。一般来说,某个问题的解会有多种,有些好,有些差,有些介于二者之间。关注解的质量指的是寻求你所能找到的最优解。现实是有时最优解不存在,每种解法有其好的一方面,此时,需要在各种解之间做折中。

(2) 协作。在制订解题计划过程中多与人交流,对发现解中的错误或改进解都会有帮助。尽量尝试着向别人陈述你的解,与别人进行头脑风暴,用开放的心态考虑各种角度反馈回来的意见,等等,都有助于通过协作完善解题计划。

(3) 迭代。第一次得出的解题计划通常不是最好的那个。因此,在得出一个解题计划后,多重复几次制订解题计划的步骤,不断地对解进行检查和改进,弥补不足。

3.3.2 分解法

在各种解题策略中,分解法(Decomposition)是计算思维的核心概念之一。计算思维通常用于解决大型复杂问题,而由于人每次能处理的问题规模有限,对大型复杂问题进行分解是必需的。

采用分解法,将对问题进行划分,得到一组子问题,这些子问题是易于理解的,并且其解是显而易见的。通常,这种分解过程会一直持续下去,直到每个子问题的解都很简单为止。在分解过程中,子问题与原问题会形成一棵树。

计算机科学中的树结构与自然界的树是相反的——根在最顶上,向下生长。树结构一般用来组织具有层次关系的数据。树中每个节点或没有父节点,或有唯一的父节点。没有父节点的节点是唯一的,称为根节点。每个节点可以有多个子节点,没有子节点的节点称为叶节点。节点之间用线连接。图 3-3 显示了一棵描述《红楼梦》中贾家部分家族关系的树,线段代表树的边,用来表示父-子层次关系。

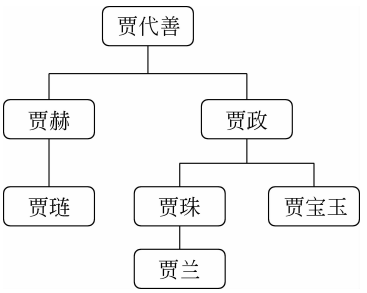


图 3-3 《红楼梦》贾家部分家族关系树

以撰写某问卷调查报告为例,来展示如何进行问题的分解。假设学校布置任务,要完成关于某主题——如共享单车——的问卷调查报告。在确定了主题后,撰写问卷调查报告不是一步就能完全解

决的问题，因为要解决该问题，所需考虑的细节太复杂，难以一次处理完，因此，需要将该问题进行分解。分解后各子问题要变得简单，通过组合各子问题的解，构成整个问题的解。

通常来说，撰写问卷调查报告须分为几个步骤：制作问卷、收集数据、分析数据、撰写报告、提交报告。据此，该问题分解为 5 个子问题，每个子问题将变得相对简单，且子问题的目标很明确。分解后得到如图 3-4 所示的树形图。

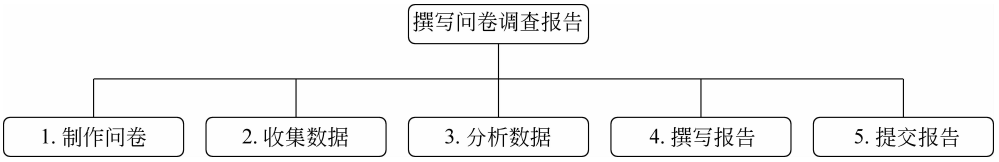


图 3-4　撰写问卷调查报告的一级任务分解图

通过分解，对撰写问卷调查这个问题认识更清楚，分解展示了解决问题所需的更多细节，也展示了更多的未知因素。例如，撰写报告子问题中，报告由哪些部分组成，格式是什么样的，目前还是未知的。此外，用什么方法收集数据，用什么工具分析数据，也是未知的，等等。

通过调研，发现有很多网站提供免费的问卷调查发布、数据统计与分析功能。假设拟基于这样的网站进行本次问卷调查报告的撰写，那么“制作问卷”任务将被分解为编写问卷题目和通过网络发布问卷两个子问题。而收集数据、分析数据将变得简单了，可直接利用网站提供的功能，在人们通过计算机、手机等方式完成问卷调查时，自动完成数据的收集和分析。因此，这两个问题将不再被分解。撰写报告任务相对而言较为复杂，可以根据报告格式分解成多个子任务，例如，撰写摘要和引言，进行数据分析等。最后提交报告只需要通过邮件将报告发送给老师即可。根据上述分析，可以得到如图 3-5 所示的问题分解树形图。

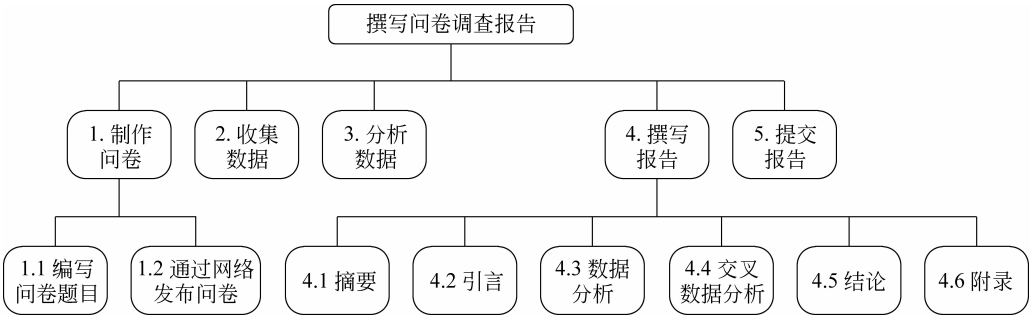


图 3-5　撰写问卷调查报告的二级任务分解图

在问题分解时，按照子问题编号逐个考察，看该问题的解是否显而易见、是否足够简单。如果是，则不需要再分解了，否则，可进一步分解成更小的子问题。该过程持续下去，直到问题分解树中所有叶节点代表的问题足够简单为止。此外，在分解过程中，不要过度陷入到寻找问题的解的细节中，即如何做，而要将主要精力放在为解决该问题，需要做

什么。

2.8 节展示了如何用分解法打印月历。此处再以绘制如图 3-6 所示的脸图形为例，利用分解法，可将绘制笑脸形状的任务分解成几个简单图案的绘制，每个简单图案的绘制都是较为容易的。

图 3-6　脸图形示例

（1）作为头的一个圆。

（2）作为眼睛的两个同心圆图形（小的圆是实心的）。

（3）作为嘴的一个圆弧。

按照这个分解，绘制笑脸图形的过程如图 3-7 所示。

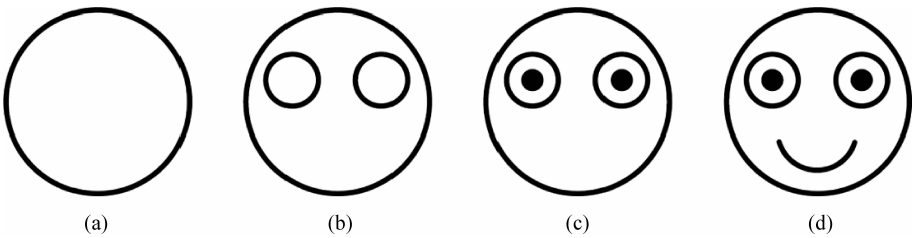


图 3-7　脸图形绘制过程

分解的结果是制订详细解题计划的起点。基于分解树，可以知道需要做什么才能解决问题，可以看到各子问题之间的依赖关系——先做什么，后做什么，哪些可以一起做。分解也有助于协作，可以将分解出来的子问题交给不同的人去（同时）完成，提高效率。

3.3.3　模式与归纳

有效的问题求解不仅仅是找到一个解决方案就结束了，还涉及在得出解后，改进解，使其更具效力。解的问题多了，会发现有些问题之间具有相似性，相似性会导致问题的解中有些元素是重复出现的，或解中有些元素是类似的。仔细考察这些相似性，找出解决方案中的相似性，归纳出普适解决方案，这是改进解决方案使其更具效力的第一步。

以图 3-7 绘制脸图形的问题为例，下面给出一个示例算法，用于绘制某个脸图案，假设初始绘制颜色默认为黑色。请注意，坐标原点在左上角，x 轴正向为右，y 轴正向为下。

1. 以坐标(50,50)为圆心，30 为半径，线宽为 2，绘制一个圆。
2. 以坐标(40,40)为圆心，6 为半径，线宽为 1，绘制一个圆。
3. 以坐标(40,40)为圆心，3 为半径，绘制一个实心圆。
4. 以坐标(60,40)为圆心，6 为半径，线宽为 1，绘制一个圆。
5. 以坐标(60,40)为圆心，3 为半径，绘制一个实心圆。
6. 线宽为 1，从坐标(30,70)到(70,70)，绘制一条红线。

这个例子中，各指令间的模式很容易归纳出来。这种归纳使得解决方案变得简单，因为包含的概念减少了。本例中，虽然有 6 步，但是只有绘制圆和绘制线两个概念。并且这两个概念可以重复应用到绘制其他图形的解决方案中。

识别解决方案中的模式有一些通用的规则，下面列举这些规则。

(1) 看看哪些名词重复出现，名词对应着解决方案的操作对象。

(2) 看看哪些动词重复出现，动词对应着解决方案的操作。

(3) 看看有哪些具体描述，这些具体描述在不同情况下可能被替换为其他的具体描述。这些可被替换的描述称为变量。例如：

① 形容词红、长、光滑表示事物的性质，可以归纳成颜色、尺寸、材质，在不同的解决方案中会被其他相应的形容词替换。

② 具体的数值，在不同的解决方案中会被其他的值替换。

根据这些原则来考察上面绘制形状的算法，可以看到：

(1) 出现的名词：圆、半径、圆心、线宽、线。

(2) 出现的动词：绘制。

(3) 出现的形容词：红色、黑色、实心。

(4) 出现的数值：各个坐标。

进一步考察，有些名词是算法操作的对象，如圆、线，而有些是对象的性质，如半径、圆心、线宽。因此可知，线和圆是形状的特例，“形状”是线和圆的归纳。算法中的动词只有一个，即绘制，它是该算法的核心操作。操作通常需要操作对象，对象是可变的，因此，通常以参数形式出现。在该算法中，绘制的参数是形状。

算法中的形容词除了颜色，还有实心，所谓实心，指的是用某种颜色填充某个形状。因此，又发现算法的一个操作，该操作有两个参数：颜色和形状。算法中出现的所有数值都是坐标，是操作对象的属性，不同的对象，坐标值不同。

根据上述分析，可以对该算法进行如下归纳，带括号的操作，括号中是操作的参数。

(1) 绘制(形状)：一个画给定形状的操作。

(2) 填充(形状，颜色)：一个用给定颜色涂满给定形状内部的操作。

(3) 形状：带有外部边界的对象，特殊形状有圆和线。

① 圆：具有半径、圆心和线宽属性的形状。

② 线：具有两个端点和线宽属性的形状。

至此，从一个具体形状的绘制算法出发，归纳出了其中的模式，并对其进行了简化。归纳出来的模式可用于绘制其他形状的算法。

上述对算法中模式的归纳还停留在操作和操作对象上，更复杂的模式需要拓展视野，寻找不同操作步之间的共性。这种归纳相对而言更难，并且需要更多的经验。但是这种归纳是值得的，后面可以看到这种模式的归纳带来的效力。

简而言之，更复杂模式的归纳有下面这些可遵循的原则。

(1) 操作步间的模式可归纳为循环。

(2) 操作步块间的模式可归纳为子过程(函数)。

(3) 条件和等式间的模式可归纳为规则。

1. 归纳为循环

仍以图 3-7 绘制脸图形的问题为例，考察下述算法。

1. 以坐标(40,40)为圆心,6 为半径,线宽为 1,绘制一个圆。
2. 以坐标(40,40)为圆心,3 为半径,绘制一个实心圆。
3. 以坐标(60,40)为圆心,6 为半径,线宽为 1,绘制一个圆。
4. 以坐标(60,40)为圆心,3 为半径,绘制一个实心圆。

经观察,可以发现第 3、4 步重复了第 1、2 步,唯一的差别是圆心坐标。由此,可以将这 4 步归纳为一个循环,如下所示。

1. 令坐标集合为{(40, 40), (60, 40)}。
2. 对坐标集合中的每个坐标(x, y):
   ① 以坐标(x, y)为圆心,6 为半径,线宽为 1,绘制一个圆。
   ② 以坐标(x, y)为圆心,3 为半径,绘制一个实心圆。

至此,归纳出了一个循环,循环的控制条件是坐标集合中的所有元素,每次会用集合中每个元素的值替换 x 和 y,分别执行循环中的两个绘制操作,即第一次 x=40 且 y=40,第二次 x=60 且 y=40。

2. 归纳为子过程

子过程是对操作步块间的模式进行归纳,操作步块是由多个操作步构成的结构,如上面的循环就是一个操作步块。该循环中两个绘制动作实现的功能是绘制一个眼睛,将这两个绘制动作置于循环内,是为了连续绘制两个眼睛。

假设在某次绘制形状时,需要绘制多个脸的形状,那么,绘制两个眼睛的循环将在绘制算法的多个地方出现。此时,多次出现的操作步块就会呈现出相似或相同的模式,当需要更改绘制眼睛的方法时,需要在算法的多个地方进行修改。对此可以进行改进,进一步归纳出绘制眼睛的子过程,将绘制眼睛的操作集中到该子过程内,而不需要在算法中重复出现多次具体绘制眼睛的操作。

对上面归纳出的循环进一步归纳,可以看到,半径和线宽是可变的,因此,可归纳为如下步骤。

1. 以坐标(x, y)为圆心,r1 为半径,线宽为 w,绘制一个圆。
2. 以坐标(x, y)为圆心,r2 为半径,绘制一个实心圆。

基于此,可以定义一个画眼睛的子过程。

"画眼睛"是一个子过程,参数为(x, y, r1, r2, w)。
1. 以坐标(x, y)为圆心,r1 为半径,线宽为 w,绘制一个圆。
2. 以坐标(x, y)为圆心,r2 为半径,绘制一个实心圆。

现在,当需要绘制眼睛时,只需要用特定的参数调用"画眼睛"子过程即可。例如:

1. 以 r1=6、r2=3、x=40、y=40、w=1 为参数调用"画眼睛"。
2. 以 r1=6、r2=3、x=60、y=40、w=1 为参数调用"画眼睛"。
3. 以 r1=4、r2=2、x=240、y=40、w=1 为参数调用"画眼睛"。
4. 以 r1=4、r2=2、x=250、y=40、w=1 为参数调用"画眼睛"。

可以看到，归纳出子过程后，绘制眼睛的操作集中在了子过程内，其他需要绘制眼睛的操作变成了子过程调用。并且，不同的参数可以绘制不同的眼睛。这样带来的好处是不言而喻的。

3. 归纳为规则

进一步考察“画眼睛”操作，可以发现构成眼睛的两个同心圆中，内圆半径是外圆半径的 1/2。如果对任何形状的眼睛来说，这个归纳都是成立的，那么，可进一步改进“画眼睛”子过程，如下所示。此处总结出来的半径间的关系就是一条规则。

“画眼睛”是一个子过程，参数为(x，y，r，w)。

1. 以坐标(x，y)为圆心，r 为半径，线宽为 w，绘制一个圆。

2. 以坐标(x，y)为圆心，$\frac{r}{2}$为半径，绘制一个实心圆。

规则通常以“当……”或“如果……则……”。例如，对绘制形状的问题来说，算法中有一些圆是不需要填充的，其实隐含着使用背景颜色填充这样的圆。因此，可归纳出一条规则：如果绘制的圆不是实心的，则其内部用背景的颜色填充。

3.3.4 小结

分解与归纳是相关的概念，分解将问题拆分成小问题，而归纳是将解决方案中的小步骤组合成较大的步骤。归纳的目的是改进问题的解，使其更易于处理，适用于更多的相似问题。

3.4 抽象与建模

抽象是计算思维的核心概念，对问题求解有非常大的作用，它会让人们聚焦于求解问题相关的细节，而避免无用细节的干扰。对抽象出来的结果，建模是用另一种方式对其进行描述，包括其静态的属性和动态的行为，利用前面所学的逻辑思维和算法思维，可以在建模的基础上构建出计算的解。

3.4.1 抽象

抽象(Abstraction)包含两个方面的含义：第一个方面指的是舍弃事物的非本质特征，仅保留与问题相关的本质特征；第二个方面指的是从众多的具体实例中抽取出共同的、本质性的特征。这是两种不同的操作。

抽象在计算思维中有着非常重要的地位，计算思维是“抽象的自动化执行”。

首先，计算思维涉及用计算机求解现实问题，那么，需要在计算机中构建现实问题的模型。但是，这种模型不是将现实问题原封不动地迁移到计算机内，而只能在计算机中描述现实问题。现实问题涉及的事物多且较为杂乱，包含大量的无用的、干扰性的细节。因

此，难以完全地描述现实问题，只能是将与问题求解相关的本质性细节保留下来，针对这些细节进行建模和求解问题。只有人自己对这些模型有很好的理解之后，才能通过编程等方法，指导计算机使用这些模型求解问题。

其次，从人处理事物的能力角度看，人最多只能在大脑中同时记忆并处理不超过 7 个信息。这个能力限制了人能同时处理的问题规模，因此，除了 3.3 节介绍的分解方法外，结合问题求解背景，进行必要的抽象是必要的。

1. 第一类抽象

以中学数学习题中常见的以汽车追及问题为例，其实出题人已经对实际情况进行了抽象，舍弃了与追及问题无关的汽车的特性，如汽车的颜色、汽车的排量、汽车的换挡方式等，仅留下了汽车的速度。在其他涉及汽车的问题上，可能汽车的速度将不再是本质特征，此时速度这个特征可被舍弃。可见，抽象是与场景相关的，事物的有些特征在某些场景下是无关紧要、可被舍弃的，而在某些场景下是本质性的、不可舍弃的。

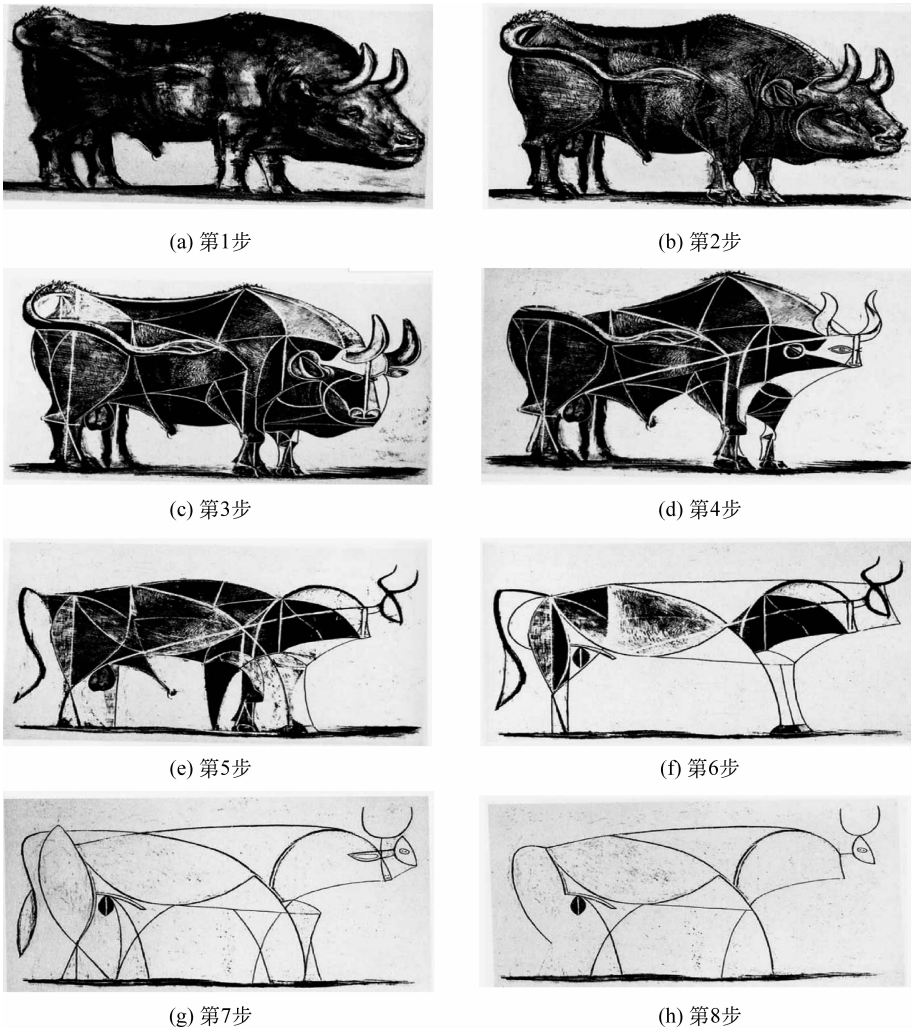
日常生活中抽象也是无处不在的，很多经常接触到的事物就是对某些真实环境的抽象。典型的例子是各城市的地铁图。美术中的抽象画派也是利用抽象这一技术，抓住绘画对象的本质特征，通过简单的线条等进行绘制。非常典型的例子是著名的抽象画派画家毕加索画牛时，只保留了“究竟哪里使得它能被我们认为是一头牛”的特征，而其他的特征被舍弃了。即如果某一个时刻，剪掉某个东西之后，它不像牛了，那么说明那个东西就是关键的、体现牛的本质特征的。

如图 3-8 所示，第 1 步的图 3-8(a)中，有许多牛的细节，比如牛皮的明暗、牛的表情、牛背上竖着的牛毛，等等。第 2 步去掉了一定的立体关系，使牛更平面，但仍然很像。第 3 步继续削弱明暗光影的因素后仍然很像牛，因此，可见光影并不是识别牛的关键。第 4 步，把牛头处理得更加抽象、简化，还是很像。第 5 步，更加简化线条，同时删去了五官。可以看到它仍然很像牛，可见五官并不是识别牛最关键的要素。第 6 步，更加简化线条，同时简化了黑白灰配比。此时仍可看出是头牛，说明黑白灰配比也不是那么重要的因素。第 7 步，彻底去掉明暗，去掉色块、黑白灰。虽然造型略抽象，但还是可以认出是牛。这表明明暗等因素都不重要。但是散乱的线条，使造型略混乱。第 8 步时，头部的眼睛等东西被简化成一个小圆圈，并把打散的线条进行了修补，使线条变的简洁，此时，比第 7 步的结果更加像头牛。

图 3-9 是毕加索画牛的最终版，可以看到，所有不关键的元素都被舍弃，留下的是牛的强壮躯干、四肢、牛角、牛尾巴，以及地面的投影。这就是一个典型的舍弃非本质特征进行抽象的例子。

2. 第二类抽象

3.3 节将在两个不同位置画同心圆得到眼睛的操作，逐步抽象成一个“画眼睛”子过程的过程，是典型的第二类抽象。通过提取画眼睛的共同特征，即除了圆心坐标不一样，其余操作都相同，将画眼睛的细节进行了屏蔽，只剩下“画眼睛”这个动作。每次调用“画眼睛”子过程就在指定的坐标处画出了眼睛，而不需要关心画眼睛具体是如何实现的。这种抽象是分层的，从高层看下去，只有“画眼睛”这个动作，而不关心细节。



(a) 第1步　(b) 第2步

(c) 第3步　(d) 第4步

(e) 第5步　(f) 第6步

(g) 第7步　(h) 第8步

图 3-8　毕加索画牛的抽象过程

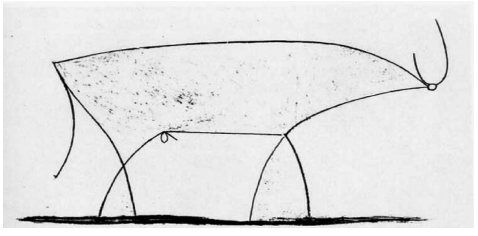


图 3-9　毕加索画的牛

继续这种抽象，可以将画脸、画眼睛、画嘴巴这些共同特征提取出来，构成“画人脸”这样一个子过程。

又例如，数学上，对数列 0、2、4、6、8、10…进行共同特征的提取，每个偶数与其出现的位置紧密相关，即第 1 个是 0、第 2 个是 2，等等。对这些共同特征进行抽象，用 n 来表示

第 n 个数，则这个数列可用 $2\times(n-1)$ 且 $n\geqslant 0$ 表示。抽象后，只看到了 $2\times(n-1)$ 这个式子，而数列中具体的偶数被屏蔽了，当需要得到第 i 个数时，只需要计算 $2\times(i-1)$ 即可。

这种类型的抽象在日常生活中是很常见的。以汽车为例，汽车是由非常复杂的机械系统、电子系统构成的。这些琐碎的复杂的机械或电子设备又分别构成点火系统、刹车系统、动力系统、转向系统等。对司机来说，开车只需要会打方向盘转向、踩油门加速、踩刹车停车、转动钥匙发动汽车即可，而不需要了解各种驾驶行为会涉及哪些复杂的机械或电子设备，以及这些设备是如何工作的。这里，点火系统、刹车系统、动力系统、转向系统等就是对复杂的汽车内机械和电子设备的一层抽象，隐藏了各系统内的复杂细节。方向盘、油门踏板、刹车踏板等是汽车提供给司机的操作界面，这个界面也是对这些系统的一层抽象，隐藏了各系统的细节。

在第 1 章中介绍了图灵机模型，它是一个抽象的计算模型。图灵把他的计算模型抽象成一种非常精简的装置：一条无限长的纸带、一个读写头、一套控制读写头工作的规则、一个状态寄存器。有了图灵机这一抽象模型，可以得到很多本质的规律，如对于计算的本质问题，计算机科学中著名的邱奇-图灵论题(The Church-Turing thesis)就说明了所有计算或算法都可以由一台图灵机来执行。本书第 5 章中介绍的冯·诺依曼体系结构就是对现代计算机体系结构的一种抽象认识。在冯·诺依曼体系结构中，计算机由内存、处理单元、控制单元、输入设备和输出设备五部分组成。这一体系结构屏蔽了实现上的诸多细节，明确了现代计算应该具备的重要组成部分及各部分之间的关系，是计算机系统的抽象模型，为现代计算机的研制奠定了基础。

网络协议也是计算机科学与技术中运用抽象思维解决复杂问题的典型。本书第 6 章介绍了网络协议的 ISO/OSI 七层体系结构模型，该模型将复杂的网络通信任务分解成 7 个层次，每个层次都是利用下一层的接口，完成本层的数据处理，并为上一层次提供更加高层的服务接口。越靠近底层的协议越接近物理实现细节，越靠近顶层的协议越接近人们的认识和理解，每一层都是在下一层的基础上做更高层的抽象，屏蔽细节，提供更高级的、更本质的服务。借助七层体系结构模型，网络系统最终完成了用户信息到物理线路信息的正确、可靠的转换，实现了计算机之间的通信。

3.4.2 建模

在讨论抽象时，经常出现的一个词是建模(modeling)，它是对现实世界事物的描述，这种描述通常会舍弃一些细节。建模的结果是各种模型，是对现实世界事物的各种表示，即抽象后的表现形式。

抽象得到的结果有很多种形式，可以是最简单的概念，如日历，就是对时间的抽象，为人们观察地球完成特定运动所需时间进行了命名，即年/月/日。又例如 3.3 节中的“画眼睛”子过程，等等。也可以是较为复杂的形式，如解数学题时画的线段图等。本节重点介绍计算思维中，设计基于计算机的解时，如何图形化地给出模型。

一般来说，模型展示了问题解决方案涉及的对象，以及对象之间的关系。并且，根据问题求解背景，所有的模型都会隐藏所建模的对象的一些细节，如图 3-10 所示。模型中