# 第5章 问题求解

在人类进行科学探索与科学研究的过程中，曾经提出过许多对科学发展具有重要影响的著名问题，如哥德巴赫猜想、庞加莱猜想、四色定理和哥尼斯堡七桥问题等，其中一些问题对计算机学科及其分支领域的形成和发展起到重要作用。另外，在计算机学科的研究工作中，为了便于理解计算机科学中有关问题和概念的本质，计算机科学家给出了不少反映该学科某一方面本质特征的典型实例，在这里称为计算机领域的典型问题。计算机领域典型问题的提出及研究，不仅有助于深刻地理解计算机学科中一些关键问题的本质，而且对学科的继续深入研究和发展具有十分重要的促进作用。本章将从图论问题、算法复杂性问题、机器智能问题、并发控制问题和分布式计算问题等进行分析。

## 5.1 问题求解的一般过程

问题求解是一个发现问题、分析问题，最后导向问题目标与结果的过程，一般包括提出问题、明确问题、提出假设、检验假设4个基本步骤。欧拉回路问题就是一个典型的问题求解过程。

18世纪中叶，东普鲁士有一座哥尼斯堡(Konigsberg)城，城中有一条贯穿全市的普雷格尔(Pregol)河，河中央有座小岛，叫奈佛夫(Kneiphof)岛，普雷格尔河的两条支流环绕其旁，并将整个城市分成北区、东区、南区和岛区4个区域，全城共有7座桥将4个城区相连起来，如图5.1所示。

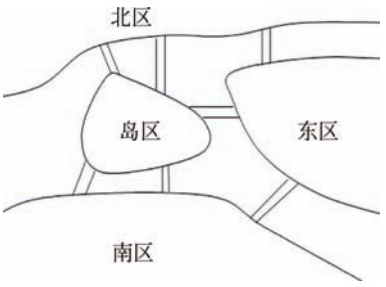

图5.1 哥尼斯堡七桥地理位置示意图

当时该城市的人们热衷一个难题：一个人怎样不重复地走完7桥，最后回到出发地点？即寻找走遍这7座桥，且每座桥只许走过一次，最后又回到原出发点的路径。所有试验者都没有解决这个难题。1736年，瑞士数学家列昂纳德·欧拉(L. Euler)发表图论的首篇论文，论证了该问题无解，即从一点出发不重复地走遍7桥，最后又回到原来出发点是不可能的。后人为了纪念数学家欧拉，将这个难题称为“哥尼斯堡七桥问题”。

为了解决哥尼斯堡七桥问题，欧拉用4个字母A、B、C和D代表4个城区，并用7条线表示7座桥，如图5.2所示，图中，只有4个点和7条边，这样做是基于该问题的本质考虑，抽象出问题最本质的东西，忽视问题非本质的东西(如桥的长度等)，从而将哥尼斯堡七桥问题抽象成为一个数学问题，即经过图中每条边一次且

仅一次的回路问题。欧拉在论文中论证了这样的回路是不存在的,后来,人们把有这样回路的图称为欧拉图,即包含有经过所有边的简单生成回路的图称为欧拉图。

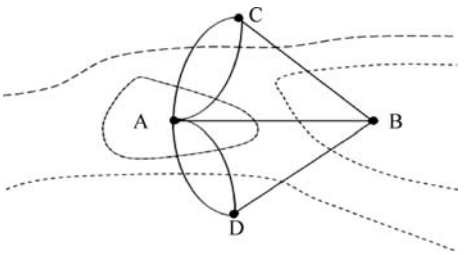


图 5.2 哥尼斯堡七桥问题示意图

欧拉在论文中将问题进行了一般化处理,即对给定的任意一个河道图与任意多座桥,判定是否可能每座桥恰好走过一次(不一定回到原出发点),并用数学方法给出了下列3条判定规则。

① 如果通奇数座桥的地方不止两个(>2),满足要求的路线是找不到的。

② 如果只有两个地方通奇数座桥,可以从这两个地方之一出发,找到所要求的路线。

③ 如果没有一个地方是通奇数座桥的,则无论从哪里出发,所要求的路线都能实现。

上述3条是欧拉通路的判定规则。

欧拉回路的判定规则如下。

图中所有地方都通偶数座桥(图中所有节点的边均为偶数)。根据判定规则可以得出,任一连通无向图[①]存在欧拉回路的充分必要条件是图的所有顶点均有偶数度。

有向欧拉通路的判定规则如下。

① 图连通。

② 除两个节点外,其余节点的入度=出度。

③ 一个节点的入度比出度大1,一个节点的入度比出度小1,或者所有节点的入度=出度。

有向欧拉回路的判定规则如下。

① 图连通。

② 所有节点的入度=出度。

欧拉的论文为图论的形成奠定了基础。今天,图论已广泛地应用于计算机科学、运筹学、信息论和控制论等学科中,并已成为人们对现实问题进行抽象的强有力的数学工具。随着计算机科学的发展,图论在计算机科学中的作用越来越大,同时,图论本身也得到了充分的发展。

## 5.2 计算机领域的典型问题

### 5.2.1 图论问题

**1. 哈密尔顿回路问题**

在图论中除了欧拉回路以外,还有一个著名的“哈密尔顿回路问题”。19世纪爱尔兰数学家哈密尔顿(Hamilton)发明了一种叫作周游世界的数学游戏。它的玩法是:给定一个正十二面体,它有20个顶点,把每个顶点看作一个城市,把正十二面体的30条棱看成连接这些城市的路。请找一条从某城市出发,经过每个城市恰好一次,并且最后回到出发点的路

① 在无向图中,每个节点连边的条数就是该节点的度数。

线。我们把正十二面体投影到平面上，在图 5.3 中标出了一种走法，即从城市 1 出发，经过 2，3，…，20，最后回到 1。

“哈密尔顿回路问题”与“欧拉回路问题”看上去十分相似，然而又是完全不同的两个问题。“哈密尔顿回路问题”是访问每个节点一次，而“欧拉回路问题”是访问每条边一次。对图是否存在“欧拉回路”前面已给出充分必要条件，而对图是否存在“哈密尔顿回路”至今仍未找到满足该问题的充分必要条件。

**2. 中国邮路问题**

我国著名数学家管梅谷教授在 1960 年提出了一个有重要理论意义和广泛应用背景的问题，被称为“中国邮路问题”。邮递员要把信送往各地点，由于送信地点多，道路不好走，还要绕过楼房，出发前需要设计一条送信路线，从邮局出发不但把信送到每个地点，而且路线不重复，最后回到邮局。这一问题可以表示为图 5.4，其中，“·”代表送信地点，空白方格“□”表示两个送信地点之间必须经过的区域，行走时不能走对角。该问题归结为图论问题就是：给定一个连通无向图（没有孤立的点），每条边都有非负的确定长度，求该图的一条经过每条边至少一次的最短回路。

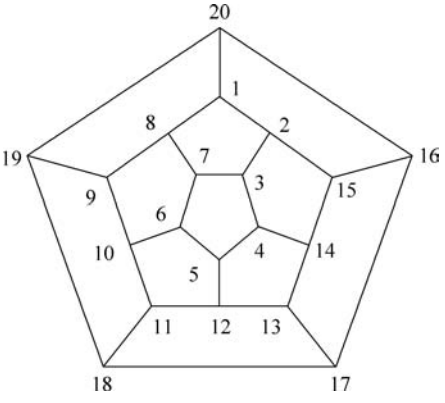


图 5.3　周游世界示意图

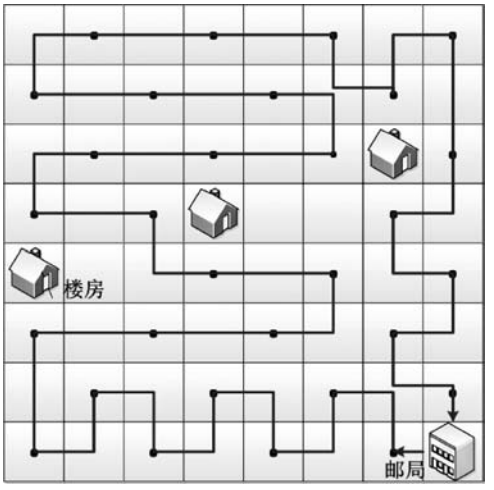


图 5.4　中国邮路问题

对于这一问题，可以采用以下几步来解决。

① 图论建模。由于街道是双向通行的，可以把它看成是赋权无向连通图，将路口抽象为点，街道抽象为边，街道的长度就是每条边的权值，问题转化为在图中求一条回路，使得回路的总权值最小。最理想的情况：若图中有欧拉回路，即可通过所有的边，因此任何一个欧拉回路即为此问题的解。

② 若无向图 G 只有两个奇点[①] $V_i$ 和 $V_j$，则有从 $V_i$ 到 $V_j$ 的欧拉迹（即欧拉通路，通过图中每条边一次且仅一次，并且过每一顶点的通路），从 $V_j$ 回到 $V_i$ 则必须重复一些边，使重复边的总长度最小，转化为求从 $V_i$ 到 $V_j$ 的最短路径。算法：找出奇点 $V_i$ 与 $V_j$ 之间的

① 在图论中，无向图 G 中，与顶点 v 关联的边的数目（环算两次），称为顶点 v 的度或次数，把度为奇数的顶点称为奇点。

最短路径P；令G′=G+P；G′为欧拉图，G′的欧拉回路即为最优邮路。

③ 一般情况，奇点数大于2时，邮路必须重复更多的边。Edmonds算法[①](匈牙利算法)思想步骤：求出G中所有奇点之间的最短路径和距离；以G的所有奇点为节点(必为偶数)，以它们之间的最短距离为节点之间边的权值，得到一个完全图$G_1$；将G的匹配图M中的匹配边$(V_i, V_j)$写成$V_i$与$V_j$之间最短路径经过的所有边的集合$E_{ij}$；令$G' = GU\{E_{ij} \mid (V_i, V_j) \in M\}$，则G′是欧拉图，求出最优邮路。

**3. 网络爬虫**

网络爬虫是一个自动提取网页的程序。整个互联网中所有网页构成的图中每个网页URL作为一个节点，网页链接构成了节点之间的边，整个万维网可以看作一个图。网络爬虫涉及图论中经典的搜索算法。

1) 广度优先搜索

广度优先搜索策略是指在网页获取过程中，在完成当前层次的搜索抓取后，再进行下一层次的搜索抓取。目前，为覆盖尽可能多的网页，一般使用广度优先搜索方法。也有很多研究将广度优先搜索策略应用于聚焦爬虫中，其基本假设是初始页面与其在一定链接距离内的网页具有主题相关性的概率很大。

2) 深度优先搜索

深度优先搜索策略从起始网页URL开始进入，分析这个网页中的其他超链接URL，选择一个URL再进入。如此一个链接一个链接地抓取下去，直到处理完一条路线之后再选择处理下一条路线。

## 5.2.2 算法复杂性问题

**1. 汉诺塔问题**

传说在古代印度的贝拿勒斯神庙里安放了一块黄铜座，座上竖有3根宝石柱子。在第一根宝石柱上，按照从小到大、自上而下的顺序放有64个直径大小不一的金盘子，形成一座金塔，如图5.5所示，即所谓的汉诺塔(又称梵天塔)。天神让庙里的僧侣们将第一根柱子上的64个盘子借助第二根柱子全部移到第三根柱子上，即将整个塔迁移，同时定下如下3条规则。

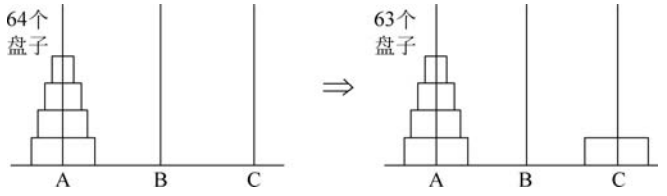


图5.5 汉诺塔问题示意图

① 每次只能移动一个盘子。

② 盘子只能在3根柱子上来回移动，不能放在别处。

① 也叫匈牙利算法，由匈牙利数学家Edmonds于1965年提出，因而得名。匈牙利算法是基于Hall定理中充分性证明的思想，它是部图匹配最常见的算法，该算法的核心就是寻找增广路径，它是一种用增广路径求二分图最大匹配的算法。

③ 在移动过程中，3 根柱子上的盘子必须始终保持大盘在下，小盘在上。

据说当这 64 个盘子全部移到第三根柱子上后，世界末日就要到了。

汉诺塔问题是一个典型的用递归方法来解决的问题。递归是计算机学科中的一个重要概念，它是将一个较大的问题归约为一个或多个子问题的求解方法，这些子问题比原问题简单，但在性质上与原问题相同。

按照这种思想要解决 64 个盘子的汉诺塔问题可以转化为 63 个盘子的汉诺塔问题。以此类推，63 个盘子的汉诺塔求解问题可以转化为 62 个盘子的汉诺塔求解问题，62 个盘子的汉诺塔求解问题又可以转化为 61 个盘子的汉诺塔求解问题，直到 1 个盘子的汉诺塔求解问题。再由 1 个盘子的汉诺塔的解求出 2 个盘子的汉诺塔……直到解出 64 个盘子的汉诺塔问题。

若从左到右的柱子依次为 A、B 和 C。移动时首先把上面 $n-1$ 个盘子移动到柱子 B 上，然后把最大的一块放在 C 上，最后把 B 上的所有盘子移动到 C 上，由此可以得出移动 $n$ 个盘子的次数 H($n$)表达式：

$$H(1)=1 \qquad (n=1) \tag{5.1}$$

$$H(n)=2H(n-1)+1 \quad (n>1) \tag{5.2}$$

那么就能得到 H($n$)的一般式(其中 $n$ 为盘子数目)：

$$H(n)=2^n-1 \quad (n>0) \tag{5.3}$$

因此，要完成 64 个盘子的汉诺塔的搬迁，需要移动盘子的次数为 $2^{64}-1=18\,446\,744\,073\,709\,551\,615$ 次。如果每次移动花费一秒，则移完这些盘子需要 5845.54 亿年以上，而地球存在至今不过 45 亿年，太阳系的预期寿命据说是数百亿年。真的过了 5845.54 亿年，不用说太阳系和银河系，至少地球上的一切生命，连同梵天塔和庙宇等，都早已经灰飞烟灭。

**2. 旅行商问题**

旅行商问题(Traveling Salesman Problem，TSP)是威廉·哈密尔顿(W. R. Hamilton)爵士和英国数学家克克曼(T. P. Kirkman)于 19 世纪初提出的一个数学问题，这是一个典型的 NP 完全性问题。其大意是：有若干个城市，任何两个城市之间的距离都是确定的，现要求一旅行商从某城市出发，必须经过每个城市且只能在每个城市逗留一次，最后回到原出发城市。问如何事先确定好一条最短的路线，使其旅行的费用最少？

人们在考虑解决这个问题时，首先想到的最原始的一种方法是：列出每条可供选择的路线(即对给定的城市进行排列组合)，计算出每条路线的总里程，最后从中选出一条最短的路线。假设现在给定 4 个城市分别为 A、B、C 和 D，各城市之间的距离为已知数，如图 5.6 和图 5.7 所示。从图中可以看到，可供选择的路线共有 6 条，可以很快选出一条总距离最短的路线。

当城市数目为 $n$ 时，那么组合路径数则为$(n-1)!$。很显然，当城市数目不多时要找到最短距离的路线并不难，但随着城市数目的不断增大，组合路线数将呈阶乘级急剧增长，以至达到无法计算的地步，这就是所谓的“组合爆炸问题”。假设现在城市的数目增为 20 个，组合路径数则为$(20-1)!\approx 1.216\times 10^{17}$，如此庞大的组合数目，若计算机以每秒检索 1000 万条路线的速度计算，也需要花上 386 年的时间。

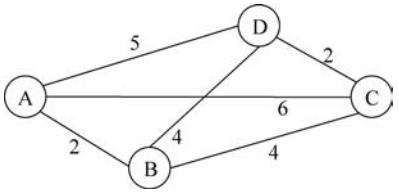


图 5.6　城市交通图

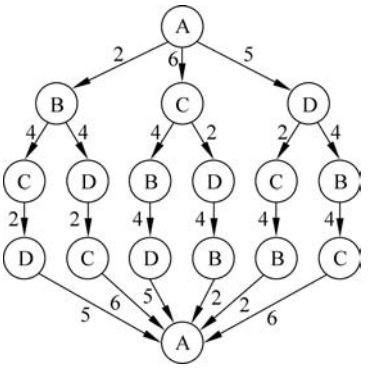


图 5.7　组合路径图

**3. 算法复杂性分析与难解性问题**

算法分析是计算机科学的一项主要工作。为了进行算法比较,必须给出算法效率的某种衡量标准。假设 $M$ 是一种算法,并设 $n$ 为输入数据的规模,实施 $M$ 所占用的时间和空间是衡量该算法效率的两个主要指标。时间由“操作”次数衡量,比如对于排序和查找,需要对比较次数计数;空间由实施该算法所需的最大内存来衡量。

算法 $M$ 的复杂性可表示为一个函数 $f(n)$,它给出了输入数据规模为 $n$ 时运行该算法所需的时间与存储空间。执行一个算法所需存储空间通常就是数据规模的倍数或平方,因此,一般“复杂性”主要指算法的运行时间。

对于时间复杂性函数 $f(n)$,它通常不仅与输入数据的规模有关,还与特定的数据有关。例如,在一篇英文短文中查找第一次出现的 3 个字母的单词 W。那么,如果 W 为定冠词 the,则 W 很可能在短文的开头部分出现,于是 $f(n)$值将会比较小;如果 W 是单词 axe,则 W 可能不会在短文中出现,则 $f(n)$会很大。因此,要考虑在适当的情况下,求出复杂性函数 $f(n)$。在复杂性理论中研究得最多的两种情况如下。

① 最坏情况。对于任何可能的输入,$f(n)$的最大值。

② 平均情况。$f(n)$的期望值。

显然,$M$ 的复杂性 $f(n)$随着 $n$ 的增大而增大。因此需要考察的是 $f(n)$的增长率,这常常由 $f(n)$与某标准函数相比较而得,例如,$\log_2 n$、$n\log_2 n$、$n^3$、$2^n$、$cn$ 等,都可用作标准函数,其中对数函数 $\log_2 n$ 增长最慢,指数函数 $2^n$ 增长最快,而多项式函数 $cn$ 的增长率随其系数 $c$ 的增大而变快。将复杂性函数与一个标准函数相比较的一种方法是利用 $O$ 标记,这里给出它的定义。

设 $f(x)$与 $g(x)$为定义于 $R$ 或 $R$ 的子集上的任意两个函数,我们说“$f(x)$与 $g(x)$同阶”,记作:

$$f(x)=O(g(x)) \tag{5.4}$$

如果存在实数 $k$ 和正常数 $C$,使得对于所有的 $x>k$,有

$$|f(x)|\leqslant C|g(x)| \tag{5.5}$$

如 $n^2+n+1\approx O(n^2)$,该表达式表示,当 $n$ 足够大时表达式左边约等于 $n^2$。

汉诺塔问题中需要移动的盘子次数为 $h(n)=2^n-1$,因此,该问题的算法时间复杂度为 $O(2^n)$。

一个算法的时间复杂度大于多项式(如指数函数)时,算法的执行时间将随 $n$ 的增加而急剧增长,以致即使是中等规模的问题也不能求解出来。于是在计算复杂性中,将这一类问题称为难解性问题,也称“NP 难解问题”。为了更好地理解计算及其复杂性的有关概念,我国学者洪加威曾经讲了一个被人称为“证比求易算法”的童话,用来帮助读者理解计算复杂性的有关概念。

很久以前,有一个年轻的国王,名叫艾述。他酷爱数学,聘请了当时最有名的数学家孔唤石当宰相。邻国有一位聪明美丽的公主,名字叫秋碧贞楠。艾述国王爱上了这位邻国公主,便亲自登门求婚。

公主说:“你如果向我求婚,请你先求出 48 770 428 433 377 171 的一个真因子,一天之内交卷。”艾述听罢,心中暗喜,心想:我从 2 开始,一个一个地试,看看能不能除尽这个数,还怕找不到这个真因子吗?艾述国王十分精于计算,他一秒钟就能算完一个数。可是,他从早到晚,共算了三万多个数,最终还是没有结果。国王向公主求情,公主将答案相告:223 092 827 是它的一个真因子。国王很快就验证了这个数确实能被 48 770 428 433 377 171 除尽。

公主说:“我再给你一次机会,如果还求不出,将来你只好做我的证婚人了”。国王立即回国,召见宰相孔唤石,大数学家在仔细地思考后认为这个数为 17 位,如果这个数可以分成两个真因子的乘积,则最小的一个真因子不会超过 9 位。于是他给国王出了一个主意:按自然数的顺序给全国的老百姓每人编一个号发下去,等公主给出数目后,立即将它们通报全国,让每个老百姓用自己的编号去除这个数,除尽了立即上报,赏黄金万两。于是,国王发动全国上下的民众,再度求婚,终于取得成功。

在“证比求易算法”的故事中,国王最先使用的是一种顺序算法,其复杂性表现在时间方面,宰相后来提出的是一种并行算法,其复杂性表现在空间方面。直觉上,我们认为顺序算法解决不了的问题完全可以用并行算法来解决,甚至会想,并行计算机系统求解问题的速度将随着处理器数目的不断增加而不断提高,从而解决难解性问题,其实这是一种误解。原因是:当将一个问题分解到多个处理器上解决时,算法中不可避免地存在必须串行执行的操作,因此会大大地限制并行计算机系统的加速能力。下面,用阿姆达尔(G. Amdahl)定律来说明这个问题。

设 $f$ 为求解某个问题的计算中存在的必须串行执行的操作占整个计算的百分比,$p$ 为处理器的数目,$Sp$ 为并行计算机系统最大的加速能力(单位:倍),则

$$Sp=\frac{1}{f+\dfrac{1-f}{p}} \tag{5.6}$$

设 $f=1\%$,$p\to\infty$,则 $Sp=100$。这说明即使在并行计算机系统中有无穷多个处理器,解决这个串行执行操作仅占全部操作 1%的问题,其解题速度与单处理器的计算机相比最多也只能提高 100 倍。因此,对难解性问题而言,单纯提高计算机系统的速度远远不够,降低算法复杂度的数量级才是最关键的。

国王有众多百姓的帮助,求亲成功是自然的事。但是,如果换成是一个平民百姓的小伙子去求婚,那就困难了。不过,小伙子可以从国王求亲成功所采用的并行算法中得到一个启发,那就是:他可以随便猜一个数,然后验证这个数。当然,这样做成功的可能性很小,不过,万一小伙子运气好猜着了呢?由于一个数和它的因子之间存在一些有规律的联系,因

此,数论知识水平较高的人猜中的可能性就大。这个小伙子使用的随机猜算法叫作非确定性算法,这样的算法需要有一种目前并不存在的非确定性计算机才能运行,其理论上的计算模型是非确定性图灵机。

在计算复杂性的研究中,所有可以在多项式时间内求解的问题称为P类问题,而所有在多项式时间内可以验证的问题称为NP类问题,P类问题采用的是确定性算法,NP类问题采用的是非确定性算法,由于确定性算法是非确定性算法的一个特例,因此P<NP。

不过,一个问题是否属于P类问题,即是否能找到这样的算法求解该问题,或证明该问题不存在这样的算法求解,至今尚未解决。20世纪70年代初,库克(S. A. Cook)和卡尔普(R. M. Karp)在P类问题是否等于NP问题上取得重大进展,指出NP类问题中有一小类问题具有以下性质:迄今为止,这些问题多数还没有人找到多项式时间计算复杂性算法。但是,一旦其中的一个问题找到了多项式时间计算复杂性算法,这个类中的其他问题也能找到多项式时间计算复杂算法,那么就可以断定P=NP。如果属于这个类中的某个问题被证明不存在多项式时间计算复杂性算法,那么,就等于证明了P≠NP。通常,将这类问题称为NP完全问题。1982年,库克因其在计算复杂性理论方面(主要是NP完全性理论方面)的奠基性工作而荣获ACM图灵奖。

随着互联网、传感器技术和通信技术的发展,数据规模快速增长,复杂性日益增强,特别是高维数据、流数据、异构数据和多模态数据等给现在的计算与问题求解提出了新的挑战。

### 5.2.3 计算智能问题

**1. 图灵测试问题**

在计算机学科诞生后,为解决人工智能中一些激烈争论的问题,图灵和西尔勒分别提出了能反映人工智能本质特征的两个著名的哲学问题,即“图灵测试”和“西尔勒中文屋子”。沿着图灵等科学家对“智能”的理解,人们在人工智能领域取得了长足的进步,其中IBM的“深蓝(Deep Blue)”战胜国际象棋大师卡斯帕罗夫(G. Kasparov)就是一个很好的例证。

图灵于1950年在英国*Mind*杂志上发表*Computing Machinery and Intelligence*一文,文中提出了“机器能思维吗?”这样一个问题,并给出了一个被后人称为“图灵测试”的模仿游戏。这个游戏由3个人来完成:一个男人($A$)、一个女人($B$)、一个性别不限的提问者($C$),提问者($C$)在与其他两个游戏者相隔离的房间里。游戏的目标是让提问者通过对其他两人的提问来鉴别其中哪个是男人,哪个是女人。为了避免提问者通过他们的声音、语调轻易地做出判断,最好是在提问者和两游戏者之间通过一台电传打字机来进行沟通。提问者只被告知两个人的代号为$X$和$Y$,游戏的最后他要做出“$X$是$A$,$Y$是$B$”或“$X$是$B$,$Y$是$A$”的判断。现在,把上面这个游戏中的男人($A$)换成一部机器来扮演,如果提问者在与机器、女人的游戏中做出的判断结果与在男人、女人之间的游戏中做出的判断结果是相同的,那么,就认为这部机器是能够思维的。

图灵关于“图灵测试”的论文发表后引发很多的关注,以后的学者在讨论机器思维时大多都要谈到这个游戏。“图灵测试”只是从功能的角度来判定机器是否能思维,也就是从行为主义角度来对机器思维进行定义。尽管图灵对机器思维的定义不够严谨,但他关于机器思维定义的开创性工作对后人的研究具有重要意义,因此,一些学者认为,图灵发表的关于“图灵测试”的论文标志着现代机器思维问题讨论的开始。

根据图灵的预测，到2000年，有机器能通过测试。现在，在某些特定的领域，如博弈领域，“图灵测试”已取得了成功，1997年，IBM公司研制的计算机“深蓝”就战胜了国际象棋冠军卡斯帕罗夫。2011年美国智力竞猜节目《危险边缘 Jeopardy!》中，IBM另一超级计算机“沃森”以3倍的分数优势夺得人机大战冠军。在未来，如果人们能像图灵揭示计算本质那样揭示人类思维的本质，即“能行”思维，那么制造真正思维机器的日子也就不远了。

**2. 西尔勒中文屋子**

一个能和人类正常交流的机器，能不能算有思想的机器呢？1980年，哲学家约翰·西尔勒(John Searle)提出了著名的“中文屋子”实验。西尔勒假设自己在一个封闭的房子里(模拟计算机)，通过有门的缝隙与外部相通，接收用中文表达的问题，但他对中文一窍不通。不过，房子里有一本英语的指令手册(相当于程序)，从中可以找到相应的回答问题的规则，他只要按照规则解答就好了。然后，把作为答案的中文符号写在纸上，送到屋子外面。这样，看起来他是能够处理中文问题的，并给出了正确答案(如同一台计算机通过了图灵测试)。但是，实际上他对那些问题毫无理解，因为他并不懂得其中的任何一个词。

西尔勒认为形式化的计算机仅有语法，只是按照规则办事，并不理解规则的含义和自己在做什么。图灵测试只是从功能角度来判定机器是否能思维。

**3. 博弈问题**

博弈问题属于人工智能中一个重要的研究领域。狭义上讲，博弈是指下棋、玩扑克牌和掷骰子等具有输赢性质的游戏；广义上讲，博弈就是对策或斗智。计算机中的博弈问题，一直是人工智能领域研究的重点内容之一。

1913年，数学家策墨洛(E. Zermelo)在第五届国际数学会议上发表《关于集合论在象棋博弈理论中的应用》(*On an Application of Set Theory to Game of Chess*)的著名论文，第一次把数学和象棋联系起来，从此，现代数学出现了一个新的理论，即博弈论。

1950年，“信息论”创始人香农(A. Shannon)发表《国际象棋与机器》(*A Chess-Playing Machine*)一文，并阐述了用计算机编制下棋程序的可能性。

1956年夏天，由麦卡锡(J. McCarthy)和香农等人共同发起的，在美国达特茅斯(Dartmouth)大学举行的夏季学术讨论会上，第一次正式使用“人工智能”这一术语，该次会议的召开对人工智能的发展起到极大的推动作用。当时，IBM公司的工程师塞缪尔(A. Samuel)也被邀请参加了“达特茅斯”会议，塞缪尔的研究专长正是电脑下棋。早在1952年，塞缪尔就运用博弈理论和状态空间搜索技术成功地研制了世界上第一个跳棋程序。该程序经不断地完善，于1959年击败了它的设计者塞缪尔本人；1962年，它又击败了美国一个州的冠军。

1970年开始直到1994年(1992年中断过一次)，ACM每年举办一次计算机国际象棋锦标赛，每年产生一个计算机国际象棋赛冠军，1991年，冠军由IBM的“深思Ⅱ(Deep Thought Ⅱ)”获得。ACM的这些工作极大地推动了博弈问题的深入研究，并促进人工智能领域的发展。1989年，卡斯帕罗夫首战“深思”，后者败北。1996年，在“深思”基础上研制出的“深蓝”曾再次与卡斯帕罗夫交战，并以2∶4负于对手。北京时间1997年5月初，在美国纽约公平大厦，“深蓝”与国际象棋冠军卡斯帕罗夫交战，前者以两胜一负三平战胜后者。“深蓝”是美国IBM公司研制的一台高性能并行计算机，由256(32 node×8)个专为国际象棋比赛设计的微处理器组成，据估计，该系统每秒可计算2亿步棋。

国际象棋、跳棋与围棋、中国象棋一样都属于双人完备博弈。所谓双人完备博弈就是两位选手对垒,轮流走步,其中一方完全知道另一方已经走过的棋步以及未来可能的走步,对弈的结果要么是一方赢(另一方输),要么是和局。对于任何一种双人完备博弈,都可以用一个博弈树(与或树)来描述,并通过博弈树搜索策略寻找最佳解。博弈树类似于状态图和问题求解搜索中使用的搜索树。搜索树上的每个节点对应一个棋局,树的分支表示棋的走步,根节点表示棋局的开始,叶节点表示棋局的结束。一个棋局的结果可以是赢、输或者和局。对于一个思考缜密的棋局来说,其博弈树是非常大的,就国际象棋来说,有$10^{120}$个节点(棋局总数),而对中国象棋来说,大约有$10^{160}$个节点,围棋更复杂,盘面状态达$10^{768}$。计算机要装下如此大的博弈树,并在合理的时间内进行详细的搜索是不可能的。因此,如何将搜索树修改到一个合理的范围,是值得研究的问题,"深蓝"就是这类研究的成果之一。

**4. 沃森智能**

"沃森(Watson)"(见图5.8)由90台IBM服务器和360个计算机芯片驱动组成,体积有10台普通冰箱那么大。它拥有15TB内存、2880个处理器、每秒可进行800 000亿次运算,这些服务器采用Linux操作系统。IBM为"沃森"配置的处理器是Power 7系列处理器,这是当前RISC(精简指令集计算机)架构中最强的处理器。它采用45nm工艺打造,拥有8个核心、32个线程,主频最高可达4.1GHz,其二级缓存达到了32MB,存储了大量图书、新闻和电影剧本资料、辞海、文选和《世界图书百科全书》(*World Book Encyclopedia*)等数百万份资料。每当读完问题的提示后,"沃森"就在不到3s的时间里对自己的数据库"挖地3尺",在长达2亿页的漫漫资料里搜索出来答案。

图5.8 沃森

沃森是基于IBM "DeepQA"(深度开放域问答系统工程)技术开发的。DeepQA技术可以读取数百万页文本数据,利用深度自然语言处理技术产生候选答案,根据诸多不同尺度评估那些问题。IBM研发团队为沃森开发的100多套算法可以在3s内解析问题,检索数百万条信息,然后再筛选还原成答案并输出成人类语言。每种算法都有其专门的功能,其中一种算法被称为"嵌套分解"算法,它可以将线索分解成两个不同的搜索功能。

沃森综合运用了自然语言处理、知识表示与推理、机器学习等技术,通过搜寻很多知识源,从多角度运用非常多的小算法,对各种可能的答案进行综合判断和学习。这就使得系统依赖少数知识源或少数算法的脆弱性得到了极大的降低,从而大大地提高了其性能。

## 5.2.4 并发控制问题

**1. 哲学家共餐问题**

对哲学家共餐问题可以做这样的描述(见图5.9):5个哲学家围坐在一张圆桌旁,每个人的面前摆有一碗面条,碗的两旁各摆有一只筷子。

假设哲学家的生活除了吃饭就是思考问题,而吃饭的时候需要左手拿一只筷子,右手拿一只筷子,然后开始进餐。吃完后又将筷子放回原处,继续思考问题。那么,一个哲学家的

活动进程可表示如下。

① 思考问题。

② 饿了停止思考，左手拿一只筷子(拿不到就等)。

③ 右手拿一只筷子(拿不到就等)。

④ 进餐。

⑤ 放右手筷子。

⑥ 放左手筷子。

⑦ 重新回到思考问题状态。

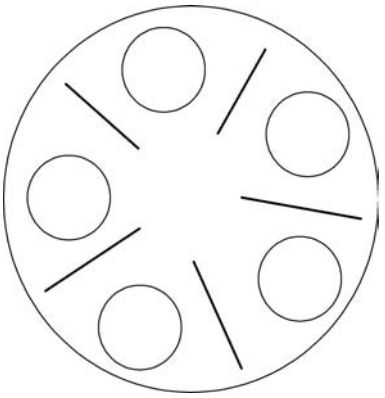

图 5.9　哲学家共餐问题

问题是：如何协调 5 个哲学家的生活进程，使得每个哲学家最终都可以进餐。考虑下面的两种情况：

按哲学家的活动进程，当所有的哲学家都同时拿起左手筷子时，则所有的哲学家都将拿不到右手的筷子，并处于等待状态，那么哲学家都将无法进餐，最终饿死。将哲学家的活动进程修改一下，变为当右手的筷子拿不到时，就放下左手的筷子，这种情况是不是就没有问题？不一定，因为可能在一个瞬间，所有的哲学家都同时拿起左手的筷子，则自然拿不到右手的筷子，于是都同时放下左手的筷子，等一会，又同时拿起左手的筷子，如此这样永远重复下去，则所有的哲学家一样都吃不到面条。

哲学家共餐问题实际上反映了计算机程序设计中多进程共享单个处理机资源时的并发控制问题。要防止这种情况发生，就必须建立一种机制，既要让每个哲学家都能吃到面条，又不能让任何一个哲学家始终拿着一根筷子不放。

以上两个方面的问题，反映的是程序并发执行时进程同步的两个问题，一个是死锁(Deadlock)，另一个是饥饿(Starvation)。与程序并发执行时进程同步有关的经典问题还有：读者-写者问题(Reader-Writer Problem)、理发师睡眠问题(Sleeping Barber Problem)等。

**2. 生产者-消费者问题**

生产者-消费者问题(Producer-consumer Problem)，也称有限缓冲问题(Bounded-buffer Problem)，是一个多线程同步问题的经典案例。该问题描述了两个共享固定大小缓冲区的线程，即所谓的“生产者”和“消费者”在实际运行时会发生的问题。生产者的主要作用是生成一定量的数据放到缓冲区中，然后重复此过程。与此同时，消费者在缓冲区消耗这些数据。该问题的关键就是要保证生产者不会在缓冲区满时加入数据，消费者也不会在缓冲区空时消耗数据。

要解决该问题，就必须让生产者在缓冲区满时休眠(要么干脆放弃数据)，等到消费者消耗了缓冲区中的数据时，生产者才能被唤醒，开始往缓冲区添加数据。同样，也可以让消费者在缓冲区空时进入休眠，等到生产者往缓冲区添加数据之后，再唤醒消费者。通常采用进程间通信的方法解决该问题，常用的方法有信号灯法等。如果解决方法不够完善，则容易出现死锁的情况。出现死锁时，两个线程都会陷入休眠，等待对方唤醒自己。该问题还被推广到多个生产者和消费者的情形。

## 5.2.5　分布式计算问题

分布式计算技术主要研究如何把一个需要巨大计算能力才能解决的问题分成许多小的部分，然后把这些部分分配给许多计算机进行处理，最后把这些计算结果综合起来得到最终

的结果。分布式计算是近年提出的一种新的计算方式。

随着计算机的普及,个人计算机开始进入千家万户,与之伴随产生的是计算机的利用问题。越来越多的计算机处于闲置状态,即使在开机状态下中央处理器的潜力也远远没有被完全利用。而且,即便是使用者实际使用计算机时,一台家用的计算机 CPU 大多数的时间处于"空闲"状态。互联网的出现,使得连接调用所有这些拥有计算资源的计算机系统成为了现实。

目前,一些本身非常复杂的但是却很适合于划分为大量的更小的计算片断的问题被提出来,然后由某个研究机构开发出基于服务端和客户端的服务程序。服务端负责将计算问题分成许多小的计算任务,然后把这些任务分配给许多联网参与计算的计算机进行并行处理,最后将这些计算结果综合起来得到最终的结果。随着参与者和参与计算的计算机的数量的不断增加,完成计算计划变得非常迅速,而且被实践证明是可行的。目前一些较大的分布式计算项目的处理能力已经可以达到或超过世界上速度最快的巨型计算机。

在分布式系统中,共享稀有资源和平衡负载是分布式计算的核心任务。负载平衡是将一个任务按一定调度算法分摊到多个操作单元上执行。建立在现有网络结构之上,它提供了一种廉价、有效、透明的方法,扩展了网络设备和服务器的带宽,增加了吞吐量,加强了网络数据的处理能力,提高了网络的灵活性和可用性,从而共同完成复杂的工作任务。

### 5.2.6 搜索排序问题

搜索是非常常见的活动,对于计算机也是一样。例如,当你在目录中查找名字时,你就在进行搜索。简单的搜索算法有顺序搜索算法和二分检索算法。顺序搜索算法遵循了搜索定义,依次查找每个元素并将其与需要搜索的元素进行比较,如果匹配,则找到了这个元素,如果不匹配,则继续找下一个元素。什么时候停止?当发现元素或者是查找所有的元素后都没有找到匹配项就停止。二分检索算法假设要检索的数组是有序的,其中每次比较操作可以找到要找的项目或把数组减少一半。二分检索不是从数组开头开始顺序前移,而是从数组中间开始。如果要搜索的项目小于数组的中间项,那么可以知道这个项目一定不会出现在数组的后半部分,因此只需要搜索数组的前半部分即可。然后再检测数组的"中间"项(即整个数组 1/4 处的项目)。如果要搜索的项目大于中间项,搜索将在数组的后半部分继续。如果中间项等于正在搜索的项目,搜索将终止。每次比较操作都会将搜索范围缩小一半。当要找的项目找到了,或可能出现这个项目的数组为空的情况,整个过程将结束。

排序是计算机内经常进行的一种操作,其目的是将一组"无序"的记录序列按照大小关系调整为"有序"的记录序列。为了简单起见,假设排序的记录都是正整数,这些正整数的个数是已知的且各不相同,要把这些正整数从小到大排序。为了求解排序问题,人们研究出了各种各样的算法,如冒泡排序、选择排序、快速排序、归并排序、堆排序等。以冒泡排序算法为例,其原理是:每次将相邻的数字进行比较,按照小的排在左边,大的排在右边进行交换。这样一趟过去后,最大的数字被交换到了最后的位置,然后再从头开始两两进行交换,直到比较到倒数第二个数时结束,以此类推。

# 习　题　5

1. 简述问题求解的基本步骤。
2. 怎样判断是否存在欧拉回路？
3. 网络爬虫涉及了图论中哪些经典算法？
4. 3 个盘子的汉诺塔从 A 柱移动到 C 柱的移动次数为多少次？
5. 何为计算的复杂性？如何衡量计算的复杂性？
6. 简要描述并发控制中的哲学家进餐问题。
7. 你在生活中遇到过什么并发控制问题，有哪些解决方法？
8. 什么是分布式计算，分布式计算的核心思想是什么？
9. 你身边的分布式计算机的例子有哪些？请举例说明。