计算机科学丛书

NPC理论导引

张泽增 著

JISUANJI KEXUE CONGSHU

贵州人民出版社

计算机科学丛书

NPC理论导引

张泽增 著

中国科学院科学基金资助的课题

封面设计　石俊生

NFC理论导引
张泽增　著
贵州人民出版社出版
（贵阳市延安中路9号）
贵州新华印刷厂印刷　贵州省新华书店经销
787×1092毫米　32开本　7.625印张　158千字
1989年3月1版　1989年3月第1次印刷
印数1—2,000
ISBN 7-221-00887-6/TP·3　定价2.46元

《计算机科学丛书》

编 委 会

编 者 的 话

为了加速发展我国的计算机科学技术，在贵州人民出版社的大力支持和协助下，中国科学院软件研究所、复旦大学、吉林大学、浙江大学、武汉大学、南京大学、上海交通大学、山东大学、哈尔滨科技大学、西北电讯工程学院、贵州大学等有关方面的同志经过多次磋商，组成了《计算机科学丛书》编委会。

这套《丛书》的作者，大多是长期从事计算机科学技术方面的科研、教学工作并在近几年内出国考察或学习过的中年同志。他们既有丰富的实践经验，又对国内外计算机科学的进展有比较清楚的了解。《丛书》将向读者介绍现代计算机科学方面的进展及其理论、方法和应用知识，每本书的内容也都自成体系，独立成册，集中介绍一个专题。为了便于学习，部分书后还列有少量习题，可供读者练习。在写作上，《丛书》力求做到篇幅短，内容新，重点突出，适于读者自学，并使读者在较短时间内对每一个专题的动向和发展趋势得到较为完整的了解。

这套《丛书》可作大专院校有关学科的教材和参考书。《丛书》以大学生、研究生为主要读者对象，也可供大专院校教师、研究工作者和计算机工作者参考。

我们相信，这套《丛书》的出版，将对广大读者了解和掌握计算机科学知识有所裨益。

《计算机科学丛书》

编 委 会

一九八六年一月

前　言

人们在使用计算机做事情包括求解问题时，总是希望占用尽可能少的存贮空间并用尽可能短的运算时间来完成任务。在进行软件系统设计时，总是尽可能地减少开销（即时空耗费）。通常把求解一个问题所需的基本运算次数囿界于输入量（或输入字长）的多项式的算法叫做有效算法或称为好的算法，多年来，算法设计者们碰到许多问题，始终未能设计出有效的算法。于是就反转来研究这些问题的固有复杂程度的问题，发现有的问题不可解（如停机问题等），有的问题虽可解，但很难解。对于难解的问题，经过探索，到60年代末及70年代初逐渐形成了计算复杂度理论，它的基础是NPC理论。这个理论在整个70年代取得了迅速的发展。至今它已成为计算机科学理论的基本知识，并已被广大计算机科技人员所熟悉。

本书为计算机专业的研究生及本科高年级学生而写，目的是介绍计算复杂度理论入门知识——NPC理论的基础概念和基础理论，并力求通俗，使具有离散数学及算法设计与分析基本知识的读者都可以读懂。

第1章是引论，在说明什么是‘问题’以后，举了三个具体例子：易解的、难解的和不可解的问题。接着介绍复杂度的直观知识，使读者不致一开始就碰到抽象的概念。在其后介绍图灵机和基于图灵机的复杂度概念，它是全书的基

础。进而介绍多项式变换，它是研究问题难易性的基本概念与方法。最后介绍非确定型计算及NPC概念，并证明一个种子定理——Cook定理。第2章介绍证明NPC的方法，主要有限制法、局部替换法与分量设计法。第3章讨论P类与NPC类的分界，并通过举例来讲述证明NPC问题的子问题属于P类的方法。第4章讨论对付NPC问题的办法，简要介绍近似算法、概率算法与并行算法。第5章介绍进一步的概念，包括$CO\text{-}NP$，强NPC，PSPACE，NPC的相对化，$\#PC$等。

本书不过分追求形式化的讨论，尽量写得通俗，使研究生及本科高年级学生都能读得懂。各章的习题，建议尽量地做一做，以便加深对该章内容的理解。

本书编写仓促，错误与不妥之处，请读者批评指正。

张泽墉

1988年3月

目　录

第1章　基础概念

本章在简述‘问题’及求解问题的‘算法’之后，对于易解的、难解的和不可解的问题各举一个例子，并结合着讨论复杂度的直观知识。在引进图灵机概念之后，讲述基于图灵机的复杂度概念并定义P类。在引进非确定型图灵机之后，定义NP类。最后讲述多项式变换、NPC概念和Cook定理。

§1.1　问题的难易性，复杂度的直观概念

§1.1.1　问题与算法

所谓问题是一个要求给出解答的一般性提问。它由两个要素组成。第一个要素是描述所有的参数，第二个要素是陈述解答必须满足的性质。前一个要素叫做‘具体实例’(instance)，后一个要素叫‘询问’(question)。

本书着重讨论两种形式的问题。第一种是组合优化问题(combinatorial optimization problem)，它是从可行解集中求出最优解的问题。第二种是判定问题(decision problem)，也叫做识别问题(recognization problem)，它只有两种可能的解，或者回答‘yes’，或者回答‘no’。下面以巡

回售货员问题（以下简记为TS）为例，给出它的两种形式。

例1.1 关于TS的组合优化问题形式可叙述如下：

具体实例：有限个城市的集合$C=\{C_1,C_2,\cdots,C_m\}$及每两个城市之间的距离$d(c_i,c_j)\in Z^+$，其中c_i，$c_j\in C(1\leqslant i,j\leqslant m)$。

询问：求出使得

$$\min\left[\left(\sum_{i=1}^{m-1}d(c_{\pi(i)},c_{\pi(i+1)})+d(c_{\pi(m)},c_{\pi(1)})\right)\right]$$

的城市序列$\langle c_{\pi(1)},c_{\pi(2)},\cdots,c_{\pi(m)}\rangle$，其中$\pi(1),\pi(2),\cdots,\pi(m)$是$1,2,\cdots,m$的一个全排列。

例1.2 关于TS的判定问题形式可叙述如下：

具体实例：一个有限个城市的集合$C=\{c_1,c_2,\cdots,c_m\}$，每两个城市之间的距离$d(c_i,c_j)\in Z^+$，以及界限$B\in Z^+$。

询问：是否存在一个排列$\langle c_{\pi(1)},c_{\pi(2)},\cdots,c_{\pi(m)}\rangle$使得

$$\left[\sum_{i=1}^{m-1}d(c_{\pi(i)},c_{\pi(i+1)})+d(c_{\pi(m)},c_{\pi(1)})\leqslant B?\right]$$

（若存在则回答'yes'，否则回答'no'）。

从这里可以看出如何把一个组合优化问题转化为相应的判定问题。若组合优化问题的目标函数之计算不太复杂，那么对应的判定问题就不比该组合优化问题更难。下面在第1～3章我们把主要注意力集中在判定问题上，所得结果是不难推广到相应的组合优化问题上去的。在第4章把主要注意力集中在组合优化问题上。

除了上面所说的两种形式的问题以外，还有其它形式的

问题。如求值问题（evaluation problem），它是求出解的值的问题，譬如求组合优化问题的最优解的值。还有计数问题（enumeration problem），譬如求出满足解答性质的解的数目的问题，等等。后面两种形式的问题，本书只在不多的地方涉及到。

所谓算法（alqorithm）是由明确指定操作顺序的规则所组成的若干个语句（statement 也叫做步step 或行 line）的集合，只要按照规则一步一步地执行一定数目的语句，便可求出问题的解答，通常的计算机程序都可看作是算法的表达形式。如果一个算法可被用于问题 π 的任何具体实例 α，并且保证可得到 π 的解，那么就说这个算法可解问题 π。算法的功能可用图 1-1 来说明。

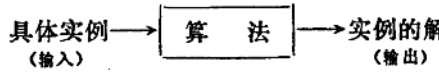

图1-1　算法的功能示意图

前面提到的关于问题的两个要素，以后对于算法来说‘具体实例’，又叫做对算法的‘输入’，也简称为实例。‘询问’即实例满足指定性质的解又叫做算法的‘输出’。为简便起见，把问题的两个要素简称为‘输入’和‘输出’。

§1.1.2　易解的问题，复杂度的直观概念

例1.3　求幂问题

输入：实数 x 和正整数 n

输出　$x^n=?$

关于求幂问题有很多算法，譬如考虑 $n=10$ 的情况，至少可

以有如下的三种方法。

（Ⅰ）计算：$x\times x\times x\times x\times x\times x\times x\times x\times x\times x\times x\times x\times x\times x$

（Ⅱ）先求出：$y=x\times x$，$z=y\times y$，$w=z\times z$，再计算：$y\times w$。

（Ⅲ）计算：$((x\times x)^2\times x)^2$

比较这三种方法的计算效率，可以对几个特定的例子将计算的时间实际测出来。但是实际测出的时间不仅跟算法的好坏有关，而且跟计算者使用计算器的熟练程度以及所用的计算器的性能都有关系。为了排除熟练程度的影响，根据在特定的计算器上按键的次数（步数）来进行比较，或者比较乘法所用的次数。后者虽然是粗略的标准，但是更具有一般性和实用性。象这样专门注意于某特定的基本步骤去数一数完成计算过程所需的总步数，就叫做这个计算过程（或算法）的时间复杂度（time complexity）。若以乘法为基本步骤，则上述三个算法（Ⅰ），（Ⅱ），（Ⅲ）的时间复杂度分别是9，4，4。同样地专门注意于存贮空间的某些单元，去数一数计算过程所用存贮单元数，就叫做该计算过程（或算法）的空间复杂度（space complexity）。若以寄存一个数的寄存器作为一个单元，则算法（Ⅰ），（Ⅱ），（Ⅲ）的空间复杂度分别为2，4，2。（存贮计算过程的中间结果与最后结果的寄存器也数在内）。由这些数值可以看出，在这几个算法当中，以（Ⅲ）为最好，算法（Ⅲ）可以推广到一般的n。

算法1.1* 计算x^n的算法

1 begin {n的m位二进数表示为$[b_{m-1}b_{m-2}\cdots b_0]_2$

---

* 初接触拟ALGOL语言的读者可先阅读本章§1.1.5。

```
    其中  b_{m-1}=1, m>1}
2   y←x;
3   for  i←m-2  downto  0  do
4   begin
5   y←y*y
6   if  b_i=1  then  y←y*x
7   end
8   end
9   {y=x^n}
```

下面分析算法1.1的计算复杂度。由算法知，除最高位不计乘法次数外，从高位往低位算起，计算乘法次数的规律为逢0加1，逢1加2。

若令 $1\underbrace{0\cdots\cdots0}_{h\text{位}}$ 的乘法次数为 h，则 $T(n)\geqslant h$。

若令 $\underbrace{1\cdots\cdots1}_{h\text{位}}$ 的乘法次数为 $2h$，则 $T(n)\leqslant 2h$。

即
$$h\leqslant T(n)\leqslant 2h \tag{1-1}$$

由 n 的二进表示知

$$2^h\leqslant n\leqslant 2^h+2^{h-1}+\cdots+2+1$$
$$=\frac{2^{h+1}-1}{2-1}=2^{h+1}-1$$

于是
$$h\leqslant\log_2 n<h+1$$

即
$$\lfloor\log_2 n\rfloor=h$$

代入(1-1)得

$$\lfloor\log_2 n\rfloor\leqslant T(n)\leqslant 2\lfloor\log_2 n\rfloor$$

容易看出，若令算法1.1的空间复杂度为 $S(n)$，则有 $S(n)=2$。

在上述讨论中，跟时空复杂度有关的参数 n 叫做问题的规模或大小（size）。在方阵求积问题中方阵的行数，多项式求值问题中的多项式次数，都可以看作问题的规模。要计算，要解决问题就必须花费时间和空间。对于一个给定的算法 A，计算所需的时间 $T_A(n)$ 通常随着问题规模的增加而增加。一个问题可以有多种算法，这些算法的时间复杂度当然不一定相同。假设求解某问题的所有的算法的集合记为 $\mathscr{F}$，如果找到了一个算法 $A\in\mathscr{F}$，其时间复杂度为 $T_A(n)$，(这就给出了解决这个问题所需时间的上界)，如果求出了解决这个问题至少需要的时间 $T(n)=\min_{A\in\mathscr{F}}T_A(n)$，(解决这个问题所需时间的下界) 即 $T(n)$ 为最优算法的复杂度，那么就称 $T(n)$ 为该问题的时间复杂度。同样，类似地可以定义问题的空间复杂度 $S(n)$。问题的时空复杂度刻划了问题的复杂程度。本书以下提到复杂度或计算复杂度时，如不特别指出，均指时间复杂度。分析已知的算法时，或者尽可能精确地求出其复杂度，或者在难于精确定时以及为了方便时，就确定复杂度函数的主要项的阶 $O(f(n))$。这里 $O(f(n))$ 表示满足下述条件的函数 g：对于某常数 $c_1, c_2 (0<c_1<c_2)$，恒有 $c_1 f(n)\leqslant g(n)\leqslant c_2 f(n)$，或者对于常数 c，$|f(n)|\leqslant c|g(n)|$，前面讨论过的算法(Ⅱ)，有 $\lfloor\log_2(n)\rfloor\leqslant T_{\text{Ⅱ}}(n)\leqslant 2\lfloor\log_2 n\rfloor$，于是可记为 $T_{\text{Ⅱ}}(n)\in O(\log_2 n)$，或 $T_{\text{Ⅱ}}(n)=O(\log_2 n)$。当问题的复杂度为 $O(n^k)$ $(k\leqslant 4)$ 的程度时，就认为该问题在实用意义下可解。当算法 A 的复杂度函数 $T_A(n)$ 为 $O(p(n))$ ($p(n)$ 是多项式函数) 时，称算法 A 为多项式时间算法或有效算法，J. Edmonds 称多项式时间算法为好的算法。实际上多项式有许多好的性质。譬如多项式与多项式相加、相乘、复合都仍是多项式。更

重要的是多项式时间算法适用于计算机执行。人们把有好的算法的问题叫做是易解的（tractable）。前面讨论过的求幂x^n的问题就是易解的。求解它，除了前面提到的三种算法以外，还可以根据加法链（addition chain）即用出现过的整数用加法生成后面的整数的办法来设计其它的算法。仍以$n=10$为例（参照图1-2）。

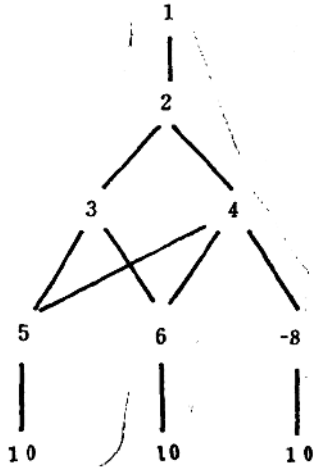

图1-2　生成整数10的加法链

前面的算法（Ⅱ）是1—2—4—5—10，即

$$x\cdot x=x^2,\ x^2\cdot x^2=x^4,\ x^4\cdot x=x^5,\ x^5\cdot x^5=x^{10}。$$

也还可以使用加法链1—2—3—5—10，即

$$x\cdot x=x^2,\ x^2\cdot x=x^3,\ x^2\cdot x^3=x^5,\ x^5\cdot x^5=x^{10}。$$

也还可以使用加法链1—2—4—8—10，即

$$x\cdot x=x^2,\ x^2\cdot x^2=x^4,\ x^4\cdot x^4=x^8,\ x^8\cdot x^2=x^{10}。$$