密码基础理论与协议

张薇 杨晓元 韩益亮 著

清华大学出版社

密码基础理论与协议

张　薇　杨晓元　韩益亮　著

清华大学出版社
北　京

内 容 简 介

本书内容涉及现代密码学的基础理论和重要协议，包括计算复杂性理论、密码函数与序列密码变换理论、典型分组密码体制、公钥密码及其安全性、数字签名、多方密码协议及可证明安全理论。

本书可作为密码学、信息安全、网络安全、电子商务等专业的科研人员、工程技术人员参考，也可供相关专业的研究生及大学本科生使用。

图书在版编目(CIP)数据

密码基础理论与协议/张薇，杨晓元，韩益亮著. —北京：清华大学出版社，2012.1
ISBN 978-7-302-26759-1

Ⅰ. ①密…　Ⅱ. ①张… ②杨… ③韩…　Ⅲ. ①密码—理论　Ⅳ. ①TN918.1

中国版本图书馆 CIP 数据核字(2011)第 184436 号

责任编辑：高买花　李　晔
责任校对：李建庄
责任印制：何　芊
出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京市清华园胶印厂
经　　销：全国新华书店
开　　本：185×260　**印　张**：14.75　**字　数**：367 千字
版　　次：2012 年 1 月第 1 版　　**印　　次**：2012 年 1 月第1 次印刷
印　　数：1～3000
定　　价：24.00 元

产品编号：039926-01

FOREWORD

前 言

本书讲述密码学的基础理论及一些重要协议，涵盖了现代密码学的大部分研究内容，因此，要求读者有一定的数学基础，包括近世代数、数论和概率论等知识。

全书共8章，第1章讲述计算复杂性理论的基本内容，使读者对算法的复杂性、问题的难度、P与NP的区别以及多项式归约的思想有一定的认识；第2章讲述密码函数及其性质，从频谱理论的角度出发，研究密码函数的相关免疫性、扩散特性、非线性性、雪崩效应、稳定性、差分特性等性能；第3章讲述序列密码，包括序列密码的基础理论、密钥序列的产生方法、序列密码的安全性及应用等内容；第4章讲述分组密码，除了介绍数据加密标准DES和高级加密标准AES这两种加密体制之外，还重点讨论了差分分析和线性分析的基本思想及方法，对于其他一些攻击分组密码的手段，如截段差分分析、高阶差分分析及非线性攻击等也作了简要介绍；第5章重点讲述近年来公钥密码研究中的热点问题，包括基于身份的公钥体制、基于格的公钥体制和椭圆曲线及超椭圆曲线密码体制等；第6章较全面地介绍了数字签名的思想及方法，包括几种经典的数字签名算法和体制以及国内外在签密方面的最新研究成果；第7章讨论多方密码协议，包括秘密共享与门限密码体制、零知识证明协议和安全多方计算等；第8章介绍密码体制的可证明安全性理论，主要内容包括形式化安全性的定义、随机预言机模型下的加密及签名方案、标准模型下的可证明安全加密方案以及数字签名的可证明安全性等。

本书是在多年教学实践和科学研究的基础上形成的，其作者一直从事密码学和信息安全方面的研究。书中的内容除在较为系统地介绍现代密码学研究的各个分支之外，还根据各位作者的研究方向在某些内容如密码函数与序列密码、新型公钥体制、数字签名与签密、安全多方计算等方面有所侧重；其中部分内容为作者的研究成果。由于作者水平有限，还有一些内容有待进一步充实，希望能在今后有机会加以弥补。

编 者

2011年12月

CONTENTS

目 录

第1章 计算复杂性理论

信息论和计算复杂性理论是现代密码学的两大基石。计算复杂性理论的核心内容是NP完全性理论，而NP完全问题是否难解是当代数学和计算机科学中尚未解决的最重要的问题之一。众所周知，公钥密码的理论基石是NP完全问题的难解性，如果对NP完全问题能找到有效解法，则绝大多数公钥密码体制将面临被攻破的威胁。本章主要介绍计算复杂性理论中最基本的内容，使读者对算法的复杂性、问题的难度、P与NP的区别以及多项式归约的思想有进一步的认识，从而更深入地理解密码体制的安全性，为后面的学习打下良好的基础。

1.1 算法的复杂性

计算复杂性理论是理论计算机科学中可计算理论的分支，它使用数学方法对计算中所需的各种资源的耗费进行定量的分析，并研究各类问题之间在计算复杂程度上的相互关系和基本性质，是算法分析的理论基础。

为了计算一类问题，总要耗费一定的时间和存储空间等资源。资源的耗费量是问题大小的函数，称为问题对该资源需求的复杂度。计算复杂性理论主要研究分析复杂度函数随问题大小而增长的阶，探讨它们对于不同的计算模型在一定意义上的无关性；根据复杂度的阶对被计算的问题分类；研究各种不同资源耗费之间的关系；估计一些基本问题的资源耗费情况的上、下界等。

计算复杂性理论中常常用到计算模型、问题、算法、时间复杂性等概念。

计算模型：为了对计算作深入的研究，需要定义一些抽象的机器，一般将这些机器称为计算模型。单带图灵机是一种最基本的计算模型，此外还有多带图灵机、随机存取机等串行计算模型和向量机等并行计算模型。

问题：需要回答的一般性提问，或者可以看做是要在计算机上求解的对象。通常一个问题含若干个参数或未给定具体取值的自由变量。

问题的描述包括两方面内容：

(1) 所有参数的一般性描述；

(2) 陈述答案或解必须满足的性质。

如果对问题中的所有未知参数指定了具体的值，就得到了该问题的一个实例。

例1-1 巡回售货员(Traveling Salesman，TS)问题。

售货员在若干个城市推销货物，已知城市间的距离，求经过所有城市的一条最短路线。

参数：城市集合 $C=\{C_1,C_2,\cdots,C_n\}$；C 中每两个城市之间的距离 $d(C_i,C_j)$，$i,j\in\{1,\cdots,n\}$。

解：这些城市的一个排列次序$\langle C_{\pi(1)},C_{\pi(2)},\cdots,C_{\pi(n)}\rangle$，使得

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

最小。其中 π 为$\{1,\cdots,n\}$上的置换。

算法：求解某个问题的一系列步骤，也可以理解为求解问题的通用程序。算法的准确性和效率是衡量算法性能的两个重要指标，此外，算法的性能还受运行范围，经济性等因素影响。算法的效率用算法在执行中所耗费的计算机资源来度量，包括时间、存储量和通信量等。

时间需求常常是决定一个具体算法是否足够高效的主要因素，因此算法的价值可以统一地用时间需求来衡量。算法的时间需求用一个函数 $T(n)$表示，其自变量 n 是问题实例的“规模”，它表示为了描述该实例所需要输入的数据总量。问题实例的规模通常用一种形式化的方式来确定，将问题看做是事先确定的一种编码方案，该编码方案将问题的实例映射到描述它们的字符串，其中的符号取自一个有穷的字符集，则问题实例的规模即为字符串的长度。

例 1-2 巡回售货员问题的一个实例。

假设共有四个城市，城市集合为$\{C_1,C_2,C_3,C_4\}$，城市间的距离如图 1-1 所示。

设字母表为：{C,[,],/,0,1,2,3,4,5,6,7,8,9}

则该实例对应的一种字符串是：C[1]C[2]C[3]C[4]//10/9/5/9/6/3，其规模为 30。

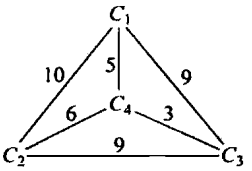

图 1-1 巡回售货员问题实例

用例 1-2 中的方法确定问题实例的规模显然是太麻烦了。问题实例的描述实际上是一种编码方式，对同一个问题实例，不同的描述方法得出的规模也不同，但它们总可以看做是问题实例输入数据长度的一个函数。因此在实际中人们会采用一种简化方法，将该问题实例输入的数据个数当做其规模。对于例 1-2 中巡回售货员问题的实例，其规模即为城市的个数 4。

如果一个算法能解答一个问题的所有实例，就说这个算法能解答这个问题。对某个问题而言，如果至少存在一个算法可以解答这个问题，就说这个问题是可解的(resolvable)，否则称这个问题为不可解的(unresolvable)。

时间复杂性：算法的时间复杂性是问题实例规模 n 的函数。对每个可能的问题实例，时间复杂性函数给出用该算法解这种规模的问题实例所需要的最长时间。

时间复杂性函数与问题的编码方案和决定着算法执行时间的计算模型有关。不同的算法具有不同的时间复杂性，根据时间复杂性可以将算法分为多项式时间算法(Polynomial Time Algorithm)和指数时间算法(Exponential Time Algorithm)。

通常用符号 O 来表示函数的数量级。对于函数 $f(x)$，如果存在常数 c 和 n_0，使得对于所有的 $n\geqslant n_0$，都有 $|f(n)|\leqslant c|g(n)|$，其中 $g(n)$是一个函数，则认为 $f(n)=O(g(n))$。例如，设 $f(n)=2n^2+7n+3$，如果取 $g(n)=n^2$，$c=3$，$n_0=8$，则当 $n\geqslant n_0$ 时，$|f(n)|\leqslant c|g(n)|$，故 $f(n)=O(n^2)$。令算法的输入长度为 n，则多项式时间算法是指时间复杂性函数为 $O(p(n))$的算法，其中 $p(n)$为 n 的多项式，设 $p(n)=a_tn^t+a_{t-1}n^{t-1}+\cdots+a_1n+a_0$，$t\in Z^+$，

则算法的时间复杂性为 $O(n^t)$，这里只保留最高次项，低次项和常数项都可忽略不计。

狭义的指数时间算法是指时间复杂性为 $O(a^{h(n)})$ 的算法，其中 a 为常量，$h(n)$ 是一个多项式，广义的指数时间算法则指除了多项式时间算法之外的所有其他算法，比如时间复杂性为 $O(n^{\log n})$ 或 $O(e^{\sqrt{n}\ln n})$ 的算法。

随着问题实例规模的 n 增大，指数时间算法所耗费的时间将呈指数递增，而多项式时间算法可以将解决问题的时间控制在合理的范围之内，所以多项式时间算法被认为是“好”的算法。表 1-1 给出了不同类型算法在相同计算条件下的运行时间。事实上，大多数指数时间算法只是穷举搜索法的变种，而多项式时间算法通常只有在对问题的结构有了某些比较深入的了解之后才能构造出来。如果一个问题不存在多项式时间算法来解决，则认为这个问题是“难解的”。

表 1-1 不同类型算法的运行时间(假设计算机执行一条指令的时间为 1μs)

算法类别		复杂性	$n=10^6$ 时的运算次数	实际运行时间
多项式时间算法	常数	$O(1)$	1	1 微秒
	线性	$O(n)$	10^6	1 秒
	二次	$O(n^2)$	10^{12}	11.6 天
	三次	$O(n^3)$	10^{18}	32 000 年
指数时间算法		$O(2^n)$	$10^{301\,030}$	约 $3\times10^{301\,016}$ 年

注：尽管指数时间算法在问题实例规模较大时会耗费难以想象的时间，但有些指数时间算法在实际中是非常有用的，这是因为时间复杂性的定义是一种最坏情况的度量，算法的时间复杂性为 2^n 仅仅表示至少有一个规模为 n 的问题实例需要这么多时间，而大多数问题实例可能需要的时间极少。比如解决线性规划问题的单纯形法虽然是指数时间算法，但在实际中它很有用处。类似的例子还有解决背包问题的分支界限法等。

对于算法设计人员来说，如果能找出各种问题相互间的联系，便可以为设计算法提供有用的信息。证明两个问题相关的基本方法是给出一个构造性变换，把第一个问题的任一实例映射到第二个问题的一个等价的实例，即把第一个问题“归约”为第二个问题，从而可以把解第二个问题的任何算法转变成解第一个问题的相应的算法。

1.2 利用 DTM 程序解决判定问题

判定问题(Decision Problem)是以“是”或“否”为解的问题。

设判定问题 Π 的全体实例集合为 D_Π，则 D_Π 可以划分为两类：Y_Π 和 N_Π，其中 Y_Π 中的实例之解均为“是”，而 N_Π 中实例之解均为“否”。

子图同构问题是图论中著名的判定问题。

例 1-3 给定两个图 $G_1=(V_1,E_1)$ 和 $G_2=(V_2,E_2)$，问 G_1 中是否包含有与 G_2 同构的子图，即是否有子集 $V'\subseteq V_1$ 和 $E'\subseteq E_1$，使得 $|V'|=|V_2|$，$|E'|=|E_2|$，并且存在一个一一映射 $f:V_2\to V'$ 满足，$(u,v)\in E_2\Leftrightarrow(f(u),f(v))\in E'$。

现实中的大多数问题，特别是优化问题，都能与判定问题建立紧密联系。如巡回售货员问题，本来是一个优化问题，但它也可以用判定问题的形式来描述：设城市集合为 $C=\{C_1,C_2,\cdots,C_n\}$，C 中每两个城市之间的距离为 $d(C_i,C_j)$，给定界限 $B\in Z^+$，问是否存在一条路线经过 C 中所有城市，且全长不超过 B，即是否存在城市的一种排列 $\langle C_{\pi(1)},C_{\pi(2)},\cdots,C_{\pi(n)}\rangle$，使得

$$\sum_{i=1}^{n-1} d(C_{\pi(i)}, C_{\pi(i+1)}) + d(C_{\pi(n)}, C_{\pi(1)}) \leqslant B$$

为了精确地研究判定问题，可以采用形式化的方法。判定问题可以用计算理论中的“语言”来形式化地表示。设 Σ 为有穷的符号集合，称为字母表，Σ^* 表示由 Σ 中所有有穷字符串组成的集合，Σ^* 的子集 L 称为字母表 Σ 上的语言。判定问题的每一个实例可以表示为字母表 Σ 上的一个有穷字符串，问题实例与字符串间的对应关系用映射 e 表示，称为该问题的编码方案。根据问题 Π 及其编码方案 e 可以将 Σ^* 中的语言划分为三类：

(1) 不是 Π 的实例；

(2) 回答为“否”的实例；

(3) 回答为“是”的实例。

第(3)类字符串称为与 Π 和 e 相关联的语言，记作

$$L[\Pi, e] = \{x \in \Sigma^* : x \text{ 是 } Y_\pi \text{ 中的一个实例在 } e \text{ 下的编码}\}$$

通过这样形式化的描述，可以将形式理论中的结论应用于判定问题，一个断言如果对于语言 $L(\Pi, e)$ 成立，则在编码方案 e 下，对问题 Π 也成立。

算法的实现须借助于计算模型。下面介绍一种最基本的计算模型，称为图灵机(Turing Machine)。图灵机是 Alan Turing 在 1936 年提出的一种假想的计算模型，它是一个具有无限读写能力的有限状态机，并且可以做无限步并行操作，是离散自动机的最一般形式。在理论上，图灵机被公认为现代计算机的原型。这台机器只保留一些最简单的指令，一个复杂的工作可以分解为这些简单指令的组合。图灵机的输入是一系列的 0 和 1，经过一组设计好的步骤，就可以解决某一特定问题。图 1-2 给出了图灵机的一般结构，其中的辅助存储器是一个具有无限容量的随机存储器。

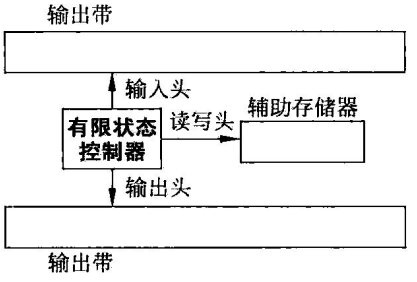

图 1-2 图灵机的一般结构

通常见到的图灵机的结构图是这个一般结构的变形：将输入带、输出带和辅助存储器画在一起，成为一个可随机读写的无限长的磁带，把输入头、输出头和读写头合并成一个具有读写功能的磁头。这种变形对自动机的功能没有带来丝毫影响，它们之间是等价的。

图灵机分为确定型和非确定型两种，确定型是指图灵机的每一步操作的结果是唯一确定的，非确定型则指图灵机的每一步操作结果及下一步操作都有多种选择，不是唯一确定的。

例 1-4 确定型单带图灵机(DTM)。

DTM 由有限状态控制器，读写头和一条磁带组成，这条磁带包含无限多个带方格，构成一个双向无穷序列，带方格被标记为…，−3，−2，−1，0，1，2，3，…(如图 1-3 所示)。

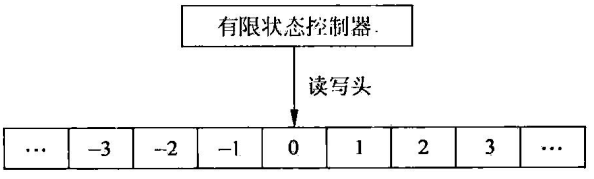

图 1-3 确定型单带图灵机

一个 DTM 程序详细规定了下述信息：

(1) 有穷的符号集合 Γ，包括输入符号子集 $\Sigma \subset \Gamma$ 和一个特殊的空白符 $b \in \Gamma - \Sigma$；

(2) 有穷的状态集合 Q，其中有一个特殊的初始状态 q_0 和两个特殊的停机状态 q_y 和 q_n；

(3) 状态转移函数 δ：$(Q-\{q_y,q_n\}) \times \Gamma \to Q \times \Gamma \times \{-1,1\}$。

DTM 的运行很简单，其输入是一个字符串 $x \in \Sigma^*$，存储在从 1 到 $|x|$ 的方格中，每一个方格放入一个符号。所有其他的方格开始时存放着空白符 b。程序从状态 q_0 开始运行，这时读写头扫描方格 1，然后计算一步一步地进行。如果当前状态 $q=q_y$ 或 q_n，那么计算结束，并且当 $q=q_y$ 时答案为“是”，当 $q=q_n$ 时答案为“否”。否则当前状态属于 $Q-\{q_y,q_n\}$，同时在被扫描的带方格中的符号 $s \in \Gamma$ 可以确定转移函数 $\delta(q,s)$ 的值。假设 $\delta(q,s)=(q',s',\Delta)$，$\Delta \in \{-1,1\}$，那么读写头擦去 s，并在这个方格里写入 s'，然后移动一格。当 $\Delta=-1$ 时向左移动一格，当 $\Delta=1$ 时向右移动一格。与此同时，有限状态控制器从状态 q 变成 q'。这就完成了一“步”计算，并为下一步计算做好了准备。

一般地，我们说输入字母表为 Σ 的 DTM 程序 M 接受输入 $x \in \Sigma^*$ 当且仅当输入为 x 时 M 停机在状态 q_y。被程序 M 识别的语言 L_M 定义为

$$L_M = \{x \in \Sigma^* : \text{M 接受 } x\}$$

语言识别的这个定义不要求 M 对 Σ^* 中的所有输入字符串都停机，而只要求对 L_M 中的输入字符串停机。如果字符串 x 属于 $\Sigma^* - L_M$，那么 M 在 x 上的计算可能停机在状态 q_n，也可能不停地进行下去。但是如果该 DTM 程序是一个算法，那么它必须对输入字母表上所有可能的字符串都停机。如果 DTM 程序 M 对输入字母表上的所有输入字符串停机并且 $L_M=L[\Pi,e]$，那么则称 M 在编码方案 e 下解判定问题 Π。

例 1-5 判定问题“整数可被 4 整除性”的 DTM 程序。

问题：给定一个正整数 N，问是否存在正整数 m，使得 $N=4m$？

已知一个正整数可被 4 整除当且仅当其二进制表示的最后二位数字是 0，根据这一点，可以用以下的 DTM 程序来解决这个问题。

$$\Gamma = \{0,1,b\}, \quad \Sigma = \{0,1\}$$

$$Q = \{q_0,q_1,q_2,q_3,q_y,q_n\}$$

状态转移函数如表 1-2 所示。

表 1-2 DTM 程序的状态转移函数

Q \ $\delta(q,s)$	0	1	b
q_0	$(q_0,0,+1)$	$(q_0,1,+1)$	$(q_1,b,-1)$
q_1	$(q_2,b,-1)$	$(q_3,b,-1)$	$(q_n,b,-1)$
q_2	$(q_y,b,-1)$	$(q_n,b,-1)$	$(q_n,b,-1)$
q_3	$(q_n,b,-1)$	$(q_n,b,-1)$	$(q_n,b,-1)$

假设输入为 10100，则程序对此输入的计算过程如表 1-3 所示。

表 1-3　DTM 程序对输入“10100”的计算过程

…	0	1	2	3	4	5	6	…
q_0	b	1*	0	1	0	0	b	…
q_0	b	1	0*	1	0	0	b	…
q_0	b	1	0	1*	0	0	b	…
q_0	b	1	0	1	0*	0	b	…
q_0	b	1	0	1	0	0*	b	…
q_0	b	1	0	1	0	0	b*	…
q_1	b	1	0	1	0	0*	b	…
q_2	b	1	0	1	0*	b	b	…
q_y	b	1	0	1*	b	b	b	…

注：* 表示读写头位置。

程序在运行 8 步之后停机在 q_y，所以对输入 10100 回答为“是”。

在例 1-5 中，被 DTM 程序识别的语言为

$$\{x \in \{0,1\}^* : x \text{ 最右边的符号至少有两个 } 0\}$$

DTM 模型是确定型算法在形式上的对应物，以上给出了一个简单的 DTM 程序的例子，实际上，DTM 程序可以完成更为复杂的工作。虽然 DTM 只有一条连续的带，在每一步也只能完成非常有限的工作，但从理论上讲，普通计算机能够完成的任何计算都可以设计 DTM 程序来完成，当然速度要慢得多。

1.3　P 与 NP

一个 DTM 程序 M 对输入 x 的计算所用的时间是指从第一步开始到进入停机状态时，在计算中出现的步数。对于对所有输入 $x \in \Sigma^*$ 都停机的 DTM 程序，其时间复杂性函数 $T_M: Z^+ \to Z^+$ 定义为

$T_M(n) = \max\{m:$ 有一个 $x \in \Sigma^*, |x| = n$，使得 M 对输入 x 的计算需要时间为 $m\}$

如果存在多项式 p，使得对于所有的 $n \in Z^+$，$T_M(n) \leqslant p(n)$，那么这样的程序 M 叫做多项式时间的 DTM 程序，它是多项式时间算法的形式对应物。

在 DTM 上可以用多项式时间解决的问题称为 P 问题：

$$P = \{L: \text{存在多项式时间的 DTM 程序 M 使得 } L = L_M\}$$

如果 $L[\Pi, e] \in P$，即存在多项式时间的 DTM 程序在编码方案 e 下“解”问题 Π，那么称判定问题 Π 在编码方案 e 下属于 P。通常不规定出具体的编码方案，而直接说判定问题 Π 属于 P。

与 P 问题相对的另一类问题称为 NP 问题。

1971 年，Stephen Cook 在其著名论文“定理证明过程的复杂性”中成功地证明了第一个 NP 完全问题，为 NP 完全性理论奠定了基础[1]。Stephen Cook 在其论文中主要做了三件重要工作。

第一，强调“多项式时间可归约性”。所谓多项式时间可归约，是指用多项式时间算法实现所需要的变换的归约，如果存在从第一个问题到第二个问题的多项式时间归约方法，则一

定可以将解决第二个问题的任何多项式时间算法转换为解决第一个问题的多项式时间算法。

第二，将讨论重点集中于 NP 类判定问题上。实际中许多非判定问题都能转化为判定问题的形式。NP 类判定问题是指可以用非确定型计算机在多项式时间内解决的问题，许多难解问题所对应的判定问题基本上都属于 NP 类判定问题。

第三，证明了 NP 类中的一个名为“可满足性”问题的具体问题具有这样的性质：NP 类中的所有其他问题都可以多项式归约为这个问题，如果可满足性问题可以用多项式时间算法解决，那么 NP 类中的所有其他问题也都可以用多项式时间算法解决。如果 NP 类中的某个问题是难解的，那么可满足性问题也一定是难解的。因此可以认为，可满足性问题是 NP 类中“最难”的问题。

第四，Cook 认为 NP 类中的一些其他问题可能具有与可满足性问题类似的性质，即也属于 NP 类中“最难”的问题。Richard Karp[2] 证明了许多著名的组合问题的判定问题形式确定恰好与可满足性问题具有相同的难度，并将这类问题定义为 NP 完全问题，它由 NP 中所有“最难”的问题组成。Karp 因此获得了 1985 年的图灵奖。Cook 的这种思想实际上将许多看上去毫不相干的复杂性问题合并为一大类，即 NP 完全问题，现在已经证明了 1000 多个不同的 NP 完全问题。

NP 完全性理论的价值在于，当我们面对一个复杂问题找不到多项式时间算法来解决时，自然会想到这个问题是否是 NP 完全问题。然而在许多情况下证明一个问题的难解性与寻找多项式时间算法的难度相近。NP 完全性理论提供了许多简单的方法来证明一个给定问题与大量的其他问题“一样的难”，而这些问题普遍被认为是很困难的。在证明某个问题是 NP 完全问题之后，就要对算法降低要求，不要去寻找有效的、精确的算法，而是针对问题的各种特殊实例来寻找有效算法，或者寻找在大多数情况下能快速运算的算法。

NP 问题可以用非确定型算法来非形式地定义，也可以用非确定型图灵机形式化地定义。

非确定型算法由两个分离的阶段组成，第一阶段为猜想阶段，第二阶段为检查阶段。给定一个问题实例 I，第一阶段仅仅猜想出某个解 S，然后，将 I 和 S 同时作为检查阶段的输入，检查阶段以普通的确定型方式进行计算，或者最后停机在答案“是”，或者停机在答案“否”，或者不停地计算下去。例如对于巡回售货员问题的判定形式，可以这样构造非确定型算法：在猜想阶段，仅仅猜测任意一个城市序列，检查阶段则验证猜测的这个城市序列间的总距离是否不超过界限 B。

例 1-6 背包问题。

给定 n 个整数的集合 $A=\{a_1,a_2,\cdots,a_n\}$ 和一个整数 M，问是否存在 A 的一个子集 A'，使得 $\sum_{x\in A'}x=M$。

这个问题可以用非确定型算法来解决，给定 A 的一个子集，验证其中元素之和是否为 M 是极其容易的，然而要寻找 A 的子集使得其中元素之和为 M，则非常困难，因为共有 2^n 个可能的子集，穷尽搜索所有子集的时间复杂性为 $O(2^n)$。

如果存在一个非确定型算法，对于问题 II 的所有实例 $I\in D_{\text{II}}$，下述两条性质成立，则称该算法“解”判定问题 II：

(1) 如果 $I \in Y_{\Pi}$，则存在 S，使得当对输入 I 猜想 S 时，检查阶段对 I 和 S 的答案为“是”；

(2) 如果 $I \notin Y_{\Pi}$，则不存在 S，使得当对输入 I 猜想 S 时，检查阶段对 I 和 S 的答案为“是”。

对于解判定问题 Π 的非确定型算法，如果存在多项式 p，使得对于每一个问题实例 $I \in Y_{\Pi}$，有猜想 S 导致确定型的检查阶段在时间 $p(\text{Length}(n))$($\text{Length}(n)$表示问题实例的规模)内对 I 和 S 回答“是”，那么称这个非确定型算法在“多项式时间”内运算。由此可以给出 NP 类的定义。

NP 类：所有可以用非确定型多项式时间算法求解的判定问题构成的类。

注意：“非确定型多项式时间算法”是根据多项式时间可验证性这一概念提出的一个术语，而不是解决判定问题的实际方法。算法对于给定的输入不是恰好有一个可能的计算，而是对每一个可能的猜想有一个可能的计算，因此有很多不同的计算。非确定型算法可以在多项式时间内检查指数种可能性。

非确定型算法在形式上的对应物是非确定型图灵机(NDTM)程序，与 DTM 程序相比，其模型中增加了一个带有只写头的猜想模块(如图 1-4 所示)。

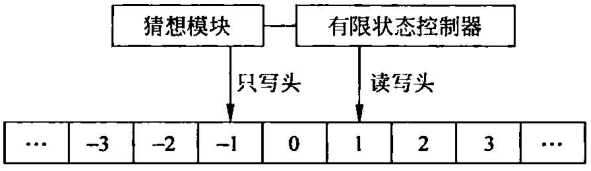

图 1-4 非确定型单带图灵机(NDTM)

假设 NDTM 程序具有带字母表 Γ，输入字母表 Σ，空白符 b，状态集合 Q，初始状态 q_0，停机状态 q_y、q_n，以及状态转移函数 δ：$(Q-\{q_y,q_n\})\times\Gamma\to Q\times\Gamma\times\{-1,+1\}$。

与非确定型算法类似，NDTM 程序在输入串 $x \in \Sigma^*$ 上的计算也可分成两个不同的阶段。第一阶段是“猜想”阶段，在带方格 1 到 $|x|$ 中写入输入字符串 x(其他所有方格是空白符)，读写头扫描带方格 1，只写头扫描带方格 −1，有限状态控制器处于“等待状态”，然后猜想模块指挥只写头，每一次一步，或者在它所扫描的那个方格里写入 Γ 中的某个符号并且向左移一格，或者停止活动。如果活动停止，此时猜想模块进入“等待状态”，而有限状态控制器变成状态 q_0 下的活动状态。如果猜想模块继续活动，则从 Γ 中任意选择一个符号再次写入。因此，猜想模块在停止活动之前，可以写下 Σ^* 中的任意字符串。

当有限状态控制器处于状态 q_0 下的活动状态时，“检查”阶段开始，从这时起，计算在 NDTM 程序的指挥下按照和 DTM 完全相同的规则进行。在检查阶段，猜想的字符串可能(通常也确实)被检查到。当且仅当有限状态控制器进入停机状态(q_y 或 q_n)时，计算停止。如果停机在状态 q_y，则这个计算称为接受计算，所有其他的计算(停机在状态 q_n 或不停机的)全都归入一类，称为拒斥计算。

任何 NDTM 程序 M 对于给定的输入字符串 x 有无穷多个可能的计算，对 Σ^* 中每一个可能的猜想字符串有一个计算。如果这些计算中至少有一个是接受计算，则称这个 NDTM 程序接受 x。可以被 M 识别的语言定义为