CAPÍTULO IV

É MAIS FÁCIL VERIFICAR A SOLUÇÃO DO QUE ENCONTRÁ-LA?

1. Introdução

No encontro da Sociedade Americana de Matemática de 1903, Frank Cole apresentou um resultado com uma característica bastante incomum. De um lado, seu resultado desprovou uma conjetura em aberto há mais de duzentos e cinqüenta anos; no entanto, não levou mais do que alguns minutos para que Cole convencesse a sua audiência de matemáticos de que a conjetura em questão era falsa, e sua prova consistia de uma única identidade:

$$2^{67} - 1 = 147.573.952.589.676.412.927 = \\ = 193.707.721 \times 761.838.257.287.$$

Evidentemente, a conjetura a que nos referimos no parágrafo anterior era a proposição de Mersenne de que números da forma $2^p - 1$ seriam primos para uma lista de certos primos p. A lista de Mersenne foi laboriosamente conferida à mão, sem o emprego de computadores, antes de 1950, sendo que o resultado de Cole foi um dos poucos erros encontrados. Desde então a lista foi consideravelmente estendida com o auxílio de computadores.

O exemplo de Cole ilustra de maneira dramática que, aparentemente, encontrar os fatores de um número composto pode em geral ser muito difícil, tanto é que levou mais de dois séculos para que alguém encontrasse os fatores de $2^{67} - 1$, mas uma vez encontrados tais fatores torna-se um problema relativamente trivial verificar que o número em questão é realmente um número composto, bastando para isso efetuar uma única multiplicação. Existe uma variedade de problemas conhecidos que parecem exibir o mesmo fenômeno, no sentido de que conhecemos algoritmos que verificam rapidamente se um canditado proposto como uma solução é ou não de fato uma solução, mas não conhecemos nenhum algoritmo rápido que encontra esta solução. Por exemplo, seja SAT o conjunto de todas as fórmulas satisfazíveis do cálculo proposicional, isto é, todas as expressões envolvendo variáveis lógicas $p, q, \ldots,$ e os operadores lógicos $\neg$ (negação), $\wedge$

(conjunção) e $\vee$ (disjunção), tais que existe uma interpretação que dê valores *verdadeiro* ou *falso* às variáveis da fórmula de tal maneira, que a fórmula com esta interpretação se torne verdadeira. Para decidir se uma fórmula F com n variáveis é satisfazível, precisamos, aparentemente, examinar 2^n possíveis interpretações. De fato não se conhece nenhum algoritmo rápido que decida se F pertence ou não a SAT. Se F for satisfazível, no entanto, uma vez conhecida a interpretação que a torne verdadeira, umas poucas operações lógicas são suficientes para verificar este fato.

Embora não se conheçam algoritmos rápidos para os problemas mencionados, tampouco conseguiu-se até agora demonstrar que tais algoritmos não existem. É então natural perguntar, intuitivamente, se verificar a solução de problemas deste tipo é inerentemente um problema mais fácil do que encontrar essa solução. Neste capítulo nos dedicaremos a formalizar esta pergunta e mostrar alguns dos resultados obtidos tentando respondê-la. Como veremos, o problema formal correspondente a nossa pergunta é uma das questões centrais em aberto em Teoria da Computação. Informalmente, o resultado principal que iremos provar afirma que existem problemas naturais de computação (SAT é um exemplo), cuja solução pode ser verificada "rapidamente" por um algoritmo, e tal que, se conseguirmos um algoritmo "rápido" para encontrar a solução de um só destes problemas, então a solução de todos os problemas que podem ser verificados rapidamente por um algoritmo pode também ser encontrada rapidamente por um algoritmo.

2. Conceitos básicos

É evidente que antes que possamos estudar a pergunta levantada informalmente na introdução é preciso caracterizar as classes de problemas cuja solução pode ser respectivamente encontrada e verificada por algoritmos rápidos.

Antes de mais nada, ocupar-nos-emos apenas com problemas que admitem por resposta *sim* ou *não*. Já examinamos alguns problemas deste tipo no Capítulo II. Restringir nossa atenção a estes problemas permite simplificar as nossas considerações e, além do mais, a maioria dos problemas de nosso interesse pode ser formulada desta maneira. Assim são, por exemplo, os problemas de determinar se um número dado é ou não composto, se uma fórmula do cálculo proposicional

é ou não satisfazível, e assim por diante. Estes problemas, portanto, equivalem a decidir se um objeto dado pertence ou não a um certo conjunto.

Em segundo lugar, podemos restringir nossa atenção a conjuntos de palavras sobre um alfabeto finito, porque qualquer objeto de nosso interesse pode ser representado por uma palavra. Por exemplo, qualquer número natural pode ser representado pela sua representação decimal que é uma palavra sobre o alfabeto $\{0, 1, 2, \ldots, 9\}$. Similarmente, qualquer fórmula proposicional pode ser vista como uma palavra sobre o alfabeto $\{p, q, \neg, \vee, \wedge, (,), v, f\}$ onde os nomes das variáveis lógicas são palavras não vazias sobre $\{p, q\}$. Na Seção 4 definiremos este conjunto com mais pormenores.

O número de símbolos do alfabeto é relativamente sem importância, desde que seja pelo menos dois, pois por uma codificação símbolo a símbolo podemos representar qualquer palavra sobre um alfabeto Σ com $m \geq 2$ símbolos, por uma palavra sobre, por exemplo, o alfabeto $\{0, 1\}$. Para isto bastarão[1] $k = \lceil log_2 m \rceil$ símbolos de $\{0, 1\}$ para representar cada símbolo de Σ, e portanto uma palavra de Σ^* de comprimento n pode ser codificada por uma palavra de $\{0, 1\}^*$ de comprimento $k \cdot n$.

Pela discussão acima, todos os problemas que nos interessarão neste capítulo serão do tipo geral de decidir se um objeto dado representado por uma palavra sobre um alfabeto conveniente pertence ou não a um certo conjunto de palavras A. Conforme vimos no Capítulo I, uma máquina de Turing determinística que reconhece A corresponde de modo natural a decidir esta questão por meio de um algoritmo, o algoritmo representado pela máquina de Turing. Similarmente, uma máquina de Turing não determinística que aceita A corresponde de modo natural a verificar, por meio de um algoritmo, se uma solução proposta ao problema correspondente a A é correta. A solução que se quer verificar, ou de maneira mais geral, informações adicionais relativas ao problema, são armazenadas na fita de sugestões. Por exemplo, para o caso dos números compostos que vimos na introdução, a fita de sugestões pode conter os fatores do número representado pela entrada. A máquina não determinística multiplicaria estes fatores e verificaria se o produto é o número representado pela entrada, aceitando se for, e rejeitando em caso contrário. É claro que tal máquina

(1) — $\lceil x \rceil$ denota o menor inteiro maior ou igual a x.

não determinística aceita o conjunto de palavras que representam números compostos, pois se a entrada representar um número composto então com uma sugestão apropriada a máquina aceita a entrada, e se a entrada não representar um número composto então esta máquina rejeitará a entrada qualquer que seja a sugestão.

3. Eficiência

Até agora não tratamos neste capítulo do problema da eficiência de nossos algoritmos. Na introdução vimos no entanto que nosso interesse é em algoritmos "rápidos".

De acordo com o Capítulo I, a complexidade de tempo associada a uma máquina de Turing determinística ou não determinística M é dada pela função $t_M(x)$, que pode ser limitada superiormente por alguma função $g : \mathbb{N} \to \mathbb{N}$. Mas qual o limite de tempo que devemos considerar "rápido"? É claro que tal pergunta só pode ser respondida num contexto empírico. Certamente tal limite deve crescer menos que uma exponencial no comprimento da entrada, pois exponenciais já crescem tão rapidamente que, se dispusermos apenas de um algoritmo exponencial para resolver um problema, então conseguiremos usá-lo apenas para entradas de tamanho relativamente reduzido. A tabela abaixo ilustra este fato. Suponhamos que dispomos de cinco algoritmos para a resolução de um certo problema, de tempos de execução n, $n \log n$, n^2, n^3 e 2^n passos respectivamente. Suponhamos ainda que dispomos de um computador que execute mil passos por segundo. A Tabela 1, fornece para cada tipo de algoritmo o comprimento máximo da entrada para vários tempos de execução, onde n representa o comprimento da entrada.

ALGORITMO	TEMPO (PASSOS)	n MÁXIMO PARA TEMPO DE EXECUÇÃO		
		1 *s*	1 min	1 h
1	n	1.000	6×10^4	$3{,}6 \times 10^6$
2	$n \log n$		4.893	$2{,}0 \times 10^5$
3	n^2	31	244	1.897
4	n^3	10	39	153
5	2^n	9	15	21

Tabela 1

Note que multiplicando o tempo disponível por uma constante multiplica o tamanho máximo da entrada por pelo menos uma constante para os primeiros quatro algoritmos, que são polinomiais, mas aumenta apenas de uma constante para o último, que é um algoritmo exponencial.

Para que consideremos um algoritmo "rápido", o limite de tempo deve então ser subexponencial. Por outro lado, para qualquer problema não trivial precisaremos certamente pelo menos tempo suficiente para ler toda a entrada, o que implica que tal limite deve ser pelo menos linear no comprimento da entrada. Uma família de limites superiores que parece razoável sob este ponto de vista é a classe de polinomiais com coeficientes inteiros não negativos $p : \mathbb{N} \to \mathbb{N}$, que denotaremos por *POL*.

A escolha de *POL* traz outras vantagens, de ordem matemática. De fato, *POL* possui propriedades de fechamento que permitem combinar algoritmos polinomiais de diversas maneiras para obter novos algoritmos polinomiais. *POL* é fechada sob soma, produto, e composição funcional. Por exemplo, um algoritmo polinomial em que cada passo e substituído por um algoritmo polinomial dá origem a um outro algoritmo polinomial.

Seja $\mathscr{P}$ a família de todos os conjuntos reconhecíveis em tempo *POL* por uma máquina de Turing determinística, e seja $\mathscr{NP}$ a família de todos os conjuntos aceitáveis em tempo *POL* por uma máquina de Turing não determinística. Pelos motivos acima parece razoável identificar a classe de problemas cuja solução pode ser encontrada rapidamente por um algoritmo com $\mathscr{P}$ e a classe de problemas cuja solução pode ser verificada rapidamente por um algoritmo com $\mathscr{NP}$. A questão levantada na introdução se traduz então na pergunta se $\mathscr{P}$ é ou não igual a $\mathscr{NP}$, ou em outras palavras, já que $\mathscr{P} \subseteq \mathscr{NP}$ pelo Exercício 1, se $\mathscr{P}$ está ou não propriamente contido em $\mathscr{NP}$. Esta questão, levantada por Cook, encontra-se ainda hoje em aberto. Nas seções seguintes veremos parte do progresso conseguido tentando resolvê-la, que ao mesmo tempo mostrará porque esta questão é um dos problemas centrais de Teoria da Computação.

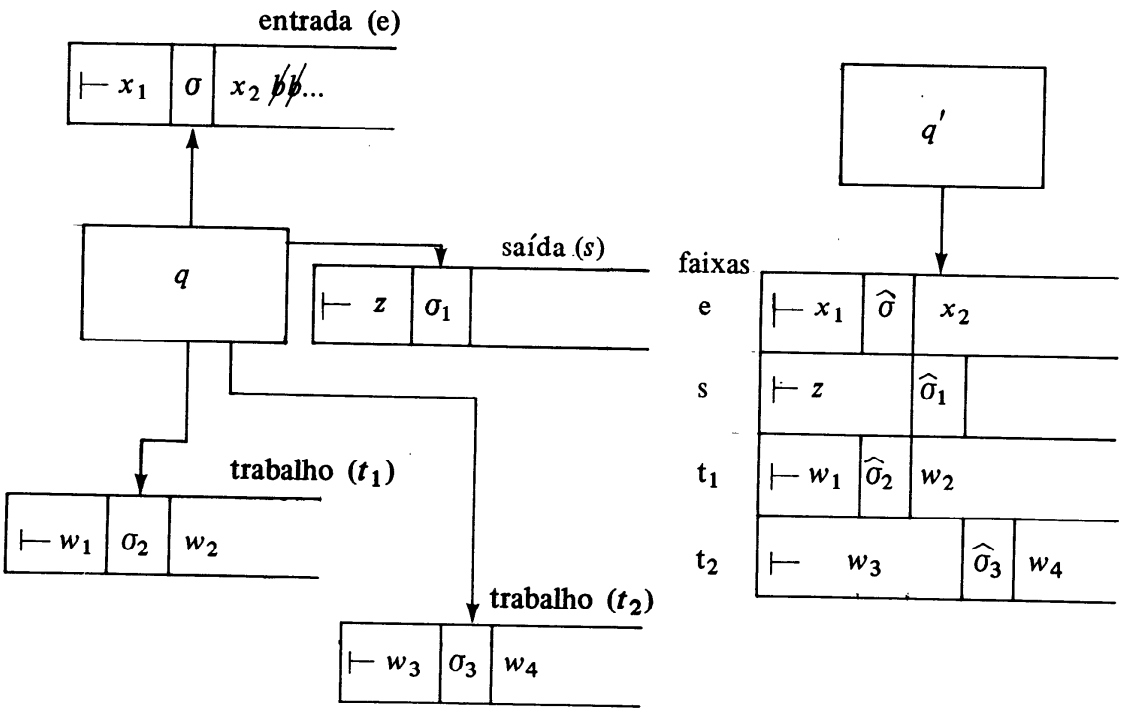
O modelo de algoritmo que definimos no Capítulo I, a máquina de Turing, é evidentemente um modelo extremamente simples. Pela Tese de Church qualquer algoritmo pode ser "programado" neste modelo. Mas será que qualquer algoritmo eficiente pode ser programado de maneira eficiente em modelo tão simples? Pareceria, à primeira vista, que poderia acontecer que uma máquina de Turing fosse ineficiente, não porque representasse um algoritmo ineficiente, mas

devido à própria simplicidade, e portanto relativa ineficiência, do modelo em si. Assim, poderíamos pensar que talvez existam algoritmos rápidos, isto é, polinomiais, em computadores reais, mas que implementados numa máquina de Turing deixariam de ser polinomiais. Um exame mais minuncioso, no entanto, revela que isto não pode acontecer. Dentre os modelos formais definidos, o mais próximo dos computadores reais é chamado *máquina de acesso aleatório*. Pode-se demonstrar que qualquer passo de uma máquina deste tipo pode ser simulado em tempo polinomial numa máquina de Turing. Para se ter uma idéia do tipo de técnicas envolvidas nestas simulações esboçamos abaixo a prova de que máquinas de Turing com várias fitas de trabalho podem ser simuladas eficientemente por uma máquina de Turing com uma única fita.

Uma *máquina de Turing com fitas múltiplas* é um sistema formal como o definido no Capítulo I, exceto que o controle central tem acesso a várias fitas de trabalho em vez de uma única fita de entrada/trabalho/saída. Normalmente neste tipo de modelo admite-se uma fita separada, de leitura apenas, contendo a entrada, e uma fita de gravação apenas, onde a máquina escreve a sua saída. Inicialmente as fitas de trabalho estão totalmente brancas. Um passo de uma máquina com fitas múltiplas é similar ao da máquina de fita única, exceto que as ações da máquina dependem agora do estado de controle central e dos símbolos lidos da fita de entrada e das fitas de trabalho. Cada cabeça das fitas de trabalho e da fita de saída escreve um símbolo na respectiva fita independentemente das demais, e cada cabeça é movida independentemente das demais no máximo uma célula para a esquerda ou para a direita na sua fita. A cabeça da fita de saída não pode ser movida para a esquerda. A função $t_M(x)$ define-se analogamente ao caso da máquina de fita única.

TEOREMA 1. *Dada uma máquina de Turing M determinística com fitas múltiplas, existe uma máquina de Turing M determinística, com uma única fita de entrada/trabalho/saída tal que (a) se M com entrada x parar em t passos então M' pára com entrada x em no máximo* $4t^2$ *passos; (b) M' aceita x sse M aceita x; (c) A saída de M' é igual à saída de M.*

Demonstração. Daremos apenas as idéias principais da demonstração. Cada uma das fitas de M é representada por uma faixa na fita única de M' conforme a Figura 1.



Máquina M com m fitas de trabalho e alfabeto $\Sigma = \{\vdash, ⊔, \sigma_0, \sigma_1, \ldots, \sigma_i\}$

Máquina M' com alfabeto $\Sigma'^{|m+2}$
$\Sigma' = \{\vdash, \hat{\vdash}, ⊔, \hat{⊔}, \sigma_0, \hat{\sigma}_0, \ldots, \sigma_i, \hat{\sigma}_i\}$ (2)

Figura 1

Um símbolo na fita de M' é uma $(m + 2)$-tupla representando o conteúdo de uma célula em cada uma das faixas correspondentes às fitas de entrada, saída e às fitas de trabalho de M. Se a cabeça numa fita de M estiver sobre esta célula então o símbolo na posição correspondente na faixa apropriada é assinalado com a marca "^". Um passo de M é simulado por M' do seguinte modo (supomos que a cabeça de M', ao se iniciar a simulação de um passo de M, encontra-se sobre a célula mais à esquerda da fita):

(a) M' desloca a cabeça para a direita até encontrar a célula em cada faixa que contém o símbolo marcado com "^". Deste modo M'

(2) Pelas convenções admitidas o alfabeto de uma máquina de Turing deve incluir pelo menos os símbolos $\{\vdash, ⊔, 0, 1\}$. Para enquadrar M' nestas convenções e para que M' calcule a mesma função que M, é preciso escolher $(m + 2)$-tuplas apropriadas para representar os símbolos de Σ. Deixamos ao leitor a implementação de pormenores deste tipo.

descobre os símbolos lidos por M neste passo e os armazena no seu controle central;

(b) Uma vez descobertos estes símbolos, M' desloca a cabeça para a esquerda escrevendo os novos símbolos escritos por M nesse passo, nas células das faixas correspondentes;

(c) Deslocando a cabeça para a direita, M' atualiza as marcas "^" das cabeças de M que se deslocam para a direita neste passo;

(d) Finalmente, deslocando a cabeça para a esquerda, M' atualiza as marcas "^" das cabeças de M que se deslocam para a esquerda neste passo, voltando também a cabeça para a célula mais à esquerda da fita, e refletindo no estado do controle central a mudança de estado de M.

Note que nenhuma cabeça de M pode afastar-se mais do que t células para a direita na respectiva fita, já que em cada passo cada cabeça pode ser deslocada de no máximo uma célula para a direita. Deste modo cada passo de M requer no máximo $4t$ passos de M' para ser simulado. Portanto a simulação toda requer no máximo $4t^2$ passos. ■

COROLÁRIO 1. *Se uma máquina de Turing M com fitas múltiplas reconhece um conjunto A em tempo polinomial, então existe uma máquina de Turing M' com uma única fita que reconhece A em tempo polinomial.*

TEOREMA 2. *Dada uma máquina de Turing não determinística M com fitas múltiplas, existe uma máquina de Turing não determinística M', com uma única fita de entrada/trabalho/saída que aceita o mesmo conjunto que M. Ademais, se M parar com entrada x e sugestão y em t passos então M' pára com entrada x e sugestão y em no máximo $4t^2$ passos.*

Demonstração. Análoga à do Teorema 1. ■

4. Fórmulas do cálculo proporcional

Vejamos agora algumas noções fundamentais de lógica que, como veremos, serão de importância para a questão $\mathscr{P} \stackrel{?}{=} \mathscr{NP}$.

Considere o alfabeto $\Sigma = \{p, q, v, f, (,), \neg, \vee, \wedge\}$.

Uma *constante proposicional* ou simplesmente *constante*, é um elemento do conjunto $\{v, f\}$. Interpretaremos v como representando *verdadeiro*, e f como representando *falso*.

Uma *variável proposicional*, ou simplesmente *variável*, é uma palavra não vazia sobre o alfabeto $\{p, q\}$. Exemplos: p, q, pq, etc.

Uma *fórmula atômica* ou *átomo* é uma variável, ou uma constante, ou uma palavra da forma $\neg w$ onde w é uma variável ou constante.

Uma *fórmula* é uma fórmula atômica ou uma palavra da forma $\neg A$, $(A \vee B)$, ou $(A \wedge B)$ onde A e B são fórmulas. Exemplos de fórmulas são: $p, f, \neg p, v, \neg v, \neg(p \wedge q), \neg(\neg\neg(p \vee q) \wedge pq)$, etc. As primeiras cinco destas fórmulas são fórmulas atômicas.

Seja A uma fórmula. Denotamos por $V(A)$ o *conjunto de variáveis* de A, definido por:

(i) Se A é uma constante então $V(A) = \varnothing$;
(ii) Se A é uma variável então $V(A) = \{A\}$;
(iii) Se A é da forma $\neg B$ onde B é uma fórmula então $V(A) = V(B)$;
(iv) Se A é da forma $(B \vee C)$ ou $(B \wedge C)$ então $V(A) = V(B) \cup V(C)$.

Seja V um conjunto de variáveis. Uma *interpretação sobre* V é uma função $I : V \to \{v, f\}$. Se $V(A) \subseteq V$ então qualquer interpretação sobre V se diz uma *interpretação de* A.

Dada uma fórmula A e uma interpretação I de A, o *valor* de A segundo I, denotado por $|| A ||_I$, é o elemento de $\{v, f\}$ definido por:

(i) Se A é uma constante então $|| A ||_I = A$;
(ii) Se A é uma variável então $|| A ||_I = I(A)$;
(iii) Se A é da forma $\neg B$ onde B é uma fórmula então

$$|| A ||_I = \begin{cases} v \text{ se } || B ||_I = f \\ f \text{ se } || B ||_I = v; \end{cases}$$

(iv) Se A é da forma $(B \vee C)$ onde B e C são fórmulas então

$$|| A ||_I = \begin{cases} v \text{ se } || B ||_I = v \text{ ou } || C ||_I = v \\ f \text{ se } || B ||_I = || C ||_I = f; \end{cases}$$

(v) Se A é da forma $(B \wedge C)$ onde B e C são fórmulas então

$$|| A ||_I = \begin{cases} v \text{ se } || B ||_I = || C ||_I = v \\ f \text{ se } || B ||_I = f \text{ ou } || C ||_I = f. \end{cases}$$

Duas fórmulas A e B são *logicamente equivalentes* se para toda interpretação I de ambas, $|| A ||_I = || B ||_I$.

Uma *disjução* de átomos (fórmulas) é uma fórmula do tipo $(((A_1 \vee A_2) \vee A_3) \vee \ldots \vee A_n)$ onde $A_1, A_2, A_3, \ldots, A_n$ são átomos (fórmulas).

Uma *conjução* de átomos (fórmulas) é uma fórmula do tipo $(((A_1 \wedge A_2) \wedge A_3) \wedge \ldots \wedge A_n)$ onde $A_1, A_2, A_3, \ldots, A_n$ são átomos (fórmulas).

Uma fórmula está em *forma normal disjuntiva* (*conjuntiva*) se for uma disjunção (conjunção) de fórmulas, cada uma das quais sendo uma conjunção (disjunção) de átomos. Exemplos: $\neg p$, $(p \vee q)$, $(p \wedge q)$, e $((p \wedge q) \vee (\neg pp \wedge pq))$ estão em forma normal disjuntiva; $\neg p$, $(p \vee q)$, $(p \wedge q)$, e $(p \wedge ((p \vee \neg q) \vee pq))$ estão em forma normal conjuntiva; $(((p \vee q) \wedge pp) \vee \neg p)$ não está nem em forma normal conjuntiva nem em forma normal disjuntiva.

Devido à propriedade associativa das operações lógicas $\wedge$ e $\vee$ (Exercício 2, (iii) e (iv)), muitos parênteses são desnecessários numa fórmula. Assim, convencionaremos, no intuito de reduzir os parênteses nas nossas expressões, que

$A_1 \vee A_2 \vee A_3 \vee \ldots \vee A_n$ abrevia $(((A_1 \vee A_2) \vee A_3) \vee \ldots \vee A_n)$,

e $A_1 \wedge A_2 \wedge A_3 \wedge \ldots \wedge A_n$ abrevia $(((A_1 \wedge A_2) \wedge A_3) \wedge \ldots \wedge A_n)$,

onde $A_1, A_2, A_3, \ldots, A_n$ são fórmulas. É também costumeiro convencionar que $\wedge$ tem precedência sobre $\vee$, e que $\neg$ tem precedência sobre $\wedge$ e $\vee$. Assim, $\neg A_1 \vee A_2$ denota $(\neg A_1 \vee A_2)$ e não $\neg(A_1 \vee A_2)$; $A_1 \vee A_2 \wedge A_3$ denota $(A_1 \vee (A_2 \wedge A_3))$ e não $((A_1 \vee A_2) \wedge A_3)$. Expressões obtidas de fórmulas, eliminando parênteses de acordo com estas regras, serão denominadas de *fórmulas abreviadas*. Seguem alguns exemplos:

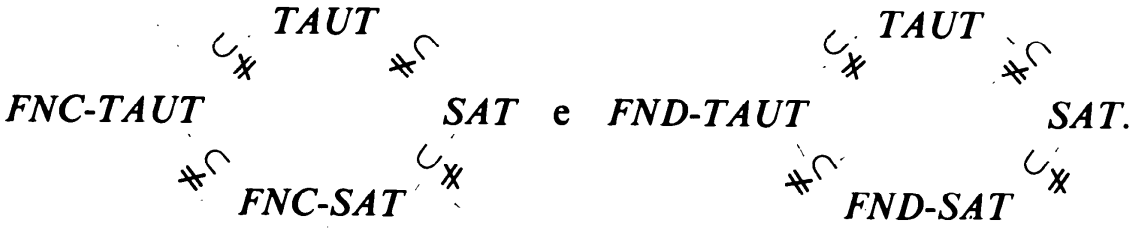
FÓRMULA	FÓRMULA ABREVIADA
$((p \wedge ((\neg p \vee q) \vee pq)) \wedge pp)$	$p \wedge (\neg p \vee q \vee pq) \wedge pp$
$\neg(A_1 \vee A_2)$	$\neg(A_1 \vee A_2)$
$(((\neg p \wedge q) \wedge pp) \vee (q \wedge pq))$	$\neg p \wedge q \wedge pp \vee q \wedge pq.$

Uma fórmula A é *satisfazível* se existe uma interpretação I de A tal que $\| A \|_I = v$.

Uma fórmula A é uma *tautologia* se for logicamente equivalente a v.

Denotaremos o conjunto de todas as fórmulas satisfazíveis, abreviadas ou não, por *SAT* e o conjunto de todas as tautologias, abreviadas ou não, por *TAUT*. *FNC-SAT* denotará o conjunto de todas as fórmulas abreviadas satisfazíveis em forma normal conjuntiva.

Analogamente, *FND-SAT* denotará o conjunto de todas as fórmulas abreviadas satisfazíveis em forma normal disjuntiva; *FNC-TAUT*, o conjunto de todas as tautologias abreviadas, em forma normal conjuntiva, e *FND-TAUT*, o conjunto de todas as tautologias abreviadas, em forma normal disjuntiva. Note que:



TEOREMA 3. *FNC-SAT pertence a* $\mathcal{NP}$.

Demonstração. Queremos construir uma máquina de Turing não determinística M que aceita *FNC-SAT* em tempo polinomial. Pelo Teorema 2 podemos supor que M tem uma fita de trabalho, e as fitas de entrada e de sugestões que são de leitura apenas. Seja x a entrada de comprimento n, e y a sugestão. A máquina M será construída de tal forma que se y for da forma $a_1b_1a_2b_2 \ldots a_mb_m$, onde $a_i \in \{p, q\}^*$, $b_i \in \{v, f\}$, $1 \leq i \leq m$, $m \leq |x| = n$ e $|y| \leq 2n$, codificando a interpretação b_i de cada variável a_i de x que torna a fórmula x *verdadeira*, e se x estiver em forma normal conjuntiva abreviada, então M aceita x. Caso contrário x é rejeitado. Note que $m \leq n$ e $|y| \leq 2n$. Para cada entrada x e sugestão y, a máquina M levará no máximo um número polinomial de passos para parar. Estas condições claramente garantem que M aceita *FNC-SAT* em tempo polinomial.

Na construção de M utilizaremos várias submáquinas com funções específicas. A máquina M será uma composição conveniente destas submáquinas. Note que uma fórmula abreviada em forma normal conjuntiva é uma expressão da forma[3]: $D_1 \wedge D_2 \wedge \ldots \wedge D_s$, onde $s \geq 1$ e cada D_i é da forma A_{1i} ou $(A_{1i} \vee A_{2i} \vee \ldots \vee A_{r_i i})$, $r_i > 1$, onde os A_{ji} são átomos. Descreveremos agora as submáquinas de M.

(3) Para simplificar a demonstração, vamos admitir a restrição adicional de que uma fórmula abreviada que consiste de uma só disjunção de vários átomos deve estar entre parênteses. Assim, a fórmula $(p \vee q)$ será aceita por M enquanto $p \vee q$ não será aceita. Note, porém, que o teorema é válido independentemente desta restrição.

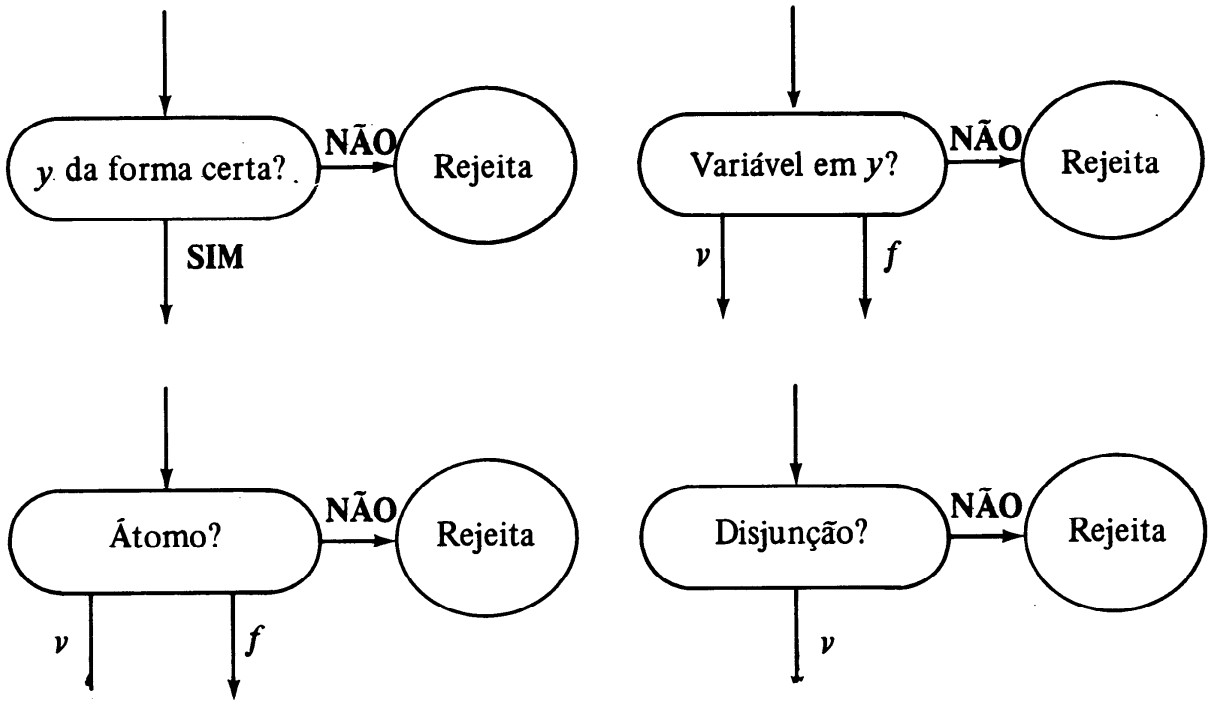


Figura 2 — As submáquinas de M.

(i) Submáquina "y é da forma certa?".

Descrição: Lê x e y simultaneamente verificando se $|y| \leq 2|x|$ e se y é da forma $a_1b_1a_2b_2 \ldots a_mb_m$ onde $a_i \in \{p, q\}^*$ e $b_i \in \{v, f\}$. Se y não for desta forma ou então se $|y| > 2|x|$ então rejeita. Caso contrário volta ambas as cabeças para o começo das respectivas fitas e devolve o comando.

Tempo de execução: $\leq 4n$.

(ii) Submáquina "Variável em y?".

Descrição: Verifica se a variável escrita na fita de trabalho ocorre em y como um dos a_i. Se não ocorrer então rejeita. Caso contrário devolve o comando no estado v se $b_i = v$, e f se $b_i = f$. A computação de "Variável em y?" começa com as cabeças da fita de sugestões e da fita de trabalho no começo das respectivas fitas, e termina com as cabeças na mesma posição. A comparação de cada a_i com a variável na fita de trabalho é feita símbolo a símbolo. Se uma das duas palavras acabar primeiro ou se algum símbolo não for o mesmo nas duas palavras então a cabeça da fita de sugestões é avançada para o próximo a_i e a cabeça da fita de trabalho é reposicionada para o símbolo mais à esquerda da variável, repetindo-se a comparação.

Tempo de execução: $\leq 2n^2 + 2n$. Cada comparação da variável na fita de trabalho com um dos a_i leva no máximo $2n$ passos, e fazemos no máximo $m \leq n$ comparações. Assim levamos no máximo $2n^2$ passos para ou encontrar a variável em y ou concluir que a mesma não ocorre em y. Levamos no máximo mais $2n$ passos para retornar as cabeças ao começo das respectivas fitas.

(iii) Submáquina "Átomo?".

Descrição: Verifica se o símbolo sob a cabeça da fita de entrada e eventualmente símbolos seguintes constituem um átomo, isto é uma seqüência de $p's$ e $q's$, ou v, ou f, ou $\neg$ seguido de uma seqüência de $p's$ e $q's$, ou $\neg v$, ou $\neg f$. Se for um átomo, devolve o valor correspondente segundo a interpretação codificada em y (para isto poderemos chamar "Variável em y?"); caso contrário, rejeita.

Tempo de execução: $\leq 2n^2 + 4n$. Leva no máximo $2n$ passos para descobrir a variável envolvida, caso haja uma, e escrevê-la na fita de trabalho retornando a cabeça da fita de trabalho ao começo da fita. Chamando "Variável em y?" levamos mais $2n^2 + 2n$ passos no máximo para descobrir o valor desta variável e conseqüentemente do átomo. Note que se a entrada for $\neg a$ então "átomo?" devolve v se o valor de a for f, e f em caso contrário.

(iv) Submáquina "Disjunção?"

Descrição: Verifica se o símbolo sob a cabeça de entrada e eventualmente símbolos seguintes constituem uma disjunção, isto é, uma palavra do tipo A_1 ou $(A_1 \vee A_2 \vee \ldots \vee A_r)$ onde $r > 1$ e A_i são átomos, e se a disjunção é verdadeira, isto é, se pelo menos um dos átomos tem valor v. Se não for, rejeita. "Disjunção?" começa verificando se o símbolo sob a cabeça é (. Se não for, então chama "Átomo?" e devolve o comando se o valor devolvido por "Átomo?" for v, e rejeita em caso contrário. Se for, então chama "Átomo?", vê se o símbolo seguinte é $\vee$, e repete este procedimento até encontrar o símbolo) em vez de $\vee$. Se qualquer dos átomos for v, devolve o comando. Caso contrário rejeita. Na Figura 3 damos o diagrama de blocos de "Disjunção?".

Tempo de execução: $\leq 2n^3 + 4n^2 + n$. Cada chamada de "Átomo?" leva no máximo $2n^2 + 4n$ passos, e mais um passo para verificar o símbolo seguinte. Há no máximo n chamadas.

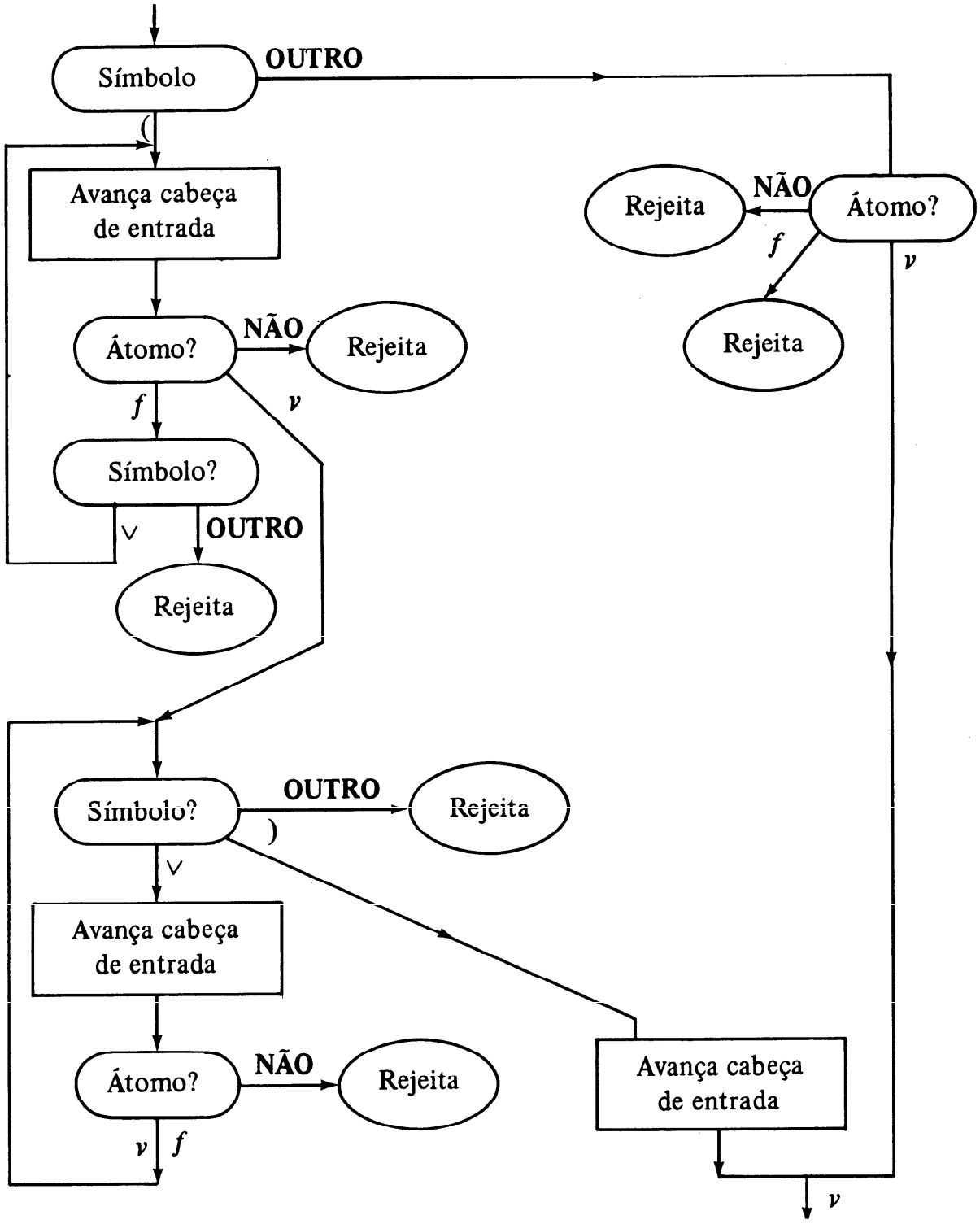


Figura 3 — Diagrama de blocos de "Disjunção?".

Com as submáquinas acima, M pode ser construido como indicado na Figura 4.

Tempo de execução: $\leq 2n^4 + 4n^3 + n^2 + 5n$. "Disjunção?" é chamada no máximo n vezes. ■

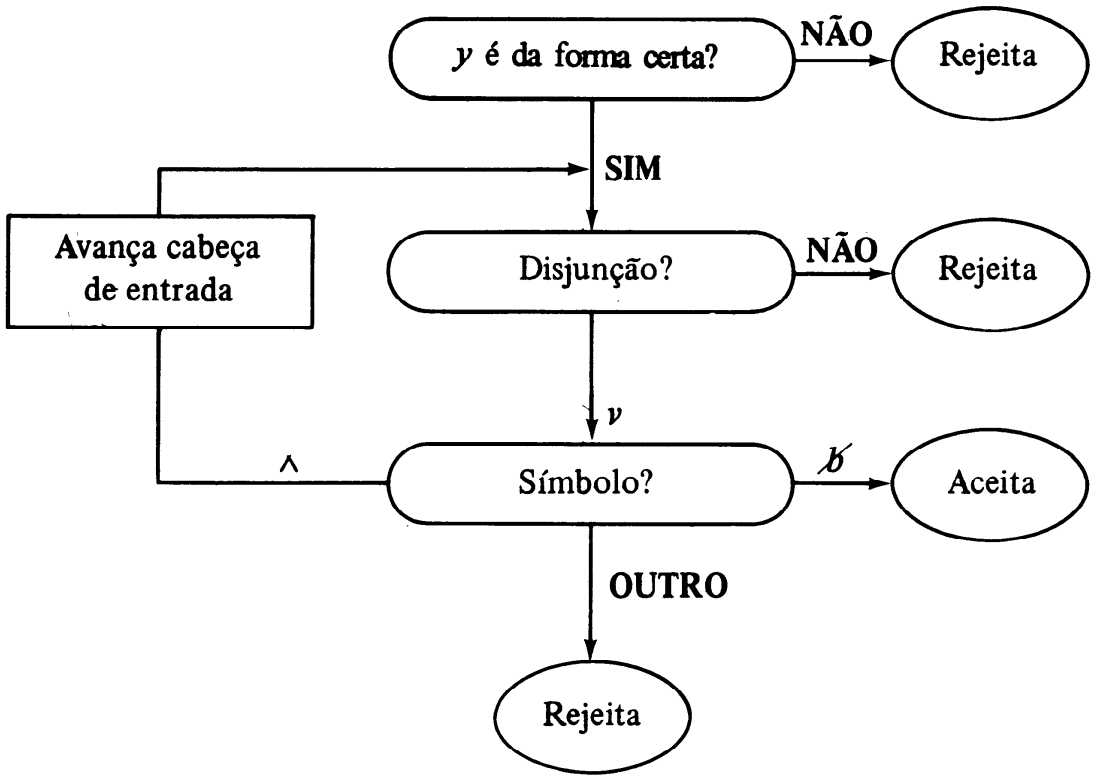


Figura 4 — Diagrama de blocos de M.

5. Redutibilidade e conjuntos NP-m-completos

No Capítulo II vimos uma noção de redutibilidade recursiva, que foi utilizada para mostrar que certos problemas são indecidíveis. Aqui introduzimos um conceito similar, o de *m*-redutibilidade polinomial.

Um conjunto $A \subseteq \Sigma^*$ é *m-redutível em tempo polinomial* a um conjunto $B \subseteq \Sigma^*$, e indicado por $A \leq_m^{\mathscr{P}} B$, se existir uma função $f : \Sigma^* \to \Sigma^*$ computável em tempo polinomial tal que para todo $x \in \Sigma^*$ temos $x \in A$ sse $f(x) \in B$.

PROPOSIÇÃO 1. *Sejam A, B e C subconjuntos de Σ^*.*
(i) *Se $A \in \mathscr{P}$ então para todo $B \subsetneq \Sigma^*$ não vazio $A \leq_m^{\mathscr{P}} B$;*
(ii) *Se $A \leq_m^{\mathscr{P}} C$ e $C \in \mathscr{P}$ então $A \in \mathscr{P}$;*
(iii) *Se $A \leq_m^{\mathscr{P}} C$ e $C \in \mathscr{NP}$ então $A \in \mathscr{NP}$.*

Demonstração. (i) Seja $b \in B$ e $b' \in \Sigma^* \setminus B$. As palavras b e b' existem porque B é não vazio e $B \subsetneq \Sigma^*$. Seja

$$f(x) = \begin{cases} b & \text{se } x \in A, \\ b' & \text{se } x \notin A. \end{cases}$$

A função f é computável em tempo polinomial porque $A \in P$, e $A \leq_m^{\mathcal{P}} B$ via f.

(ii) Se $A \leq_m^{\mathcal{P}} C$ via f então $x \in A$ sse $f(x) \in C$.

A máquina de Turing determinística M seguinte reconhece A em tempo polinomial:

(1) Calculamos $f(x)$ sobre a fita de trabalho (isto leva tempo polinomial, porque f é computável em tempo polinomial);

(2) Aplicamos a máquina de Turing que reconhece C em tempo polinomial à fita de entrada/trabalho/saída que contém $f(x)$. (Esta máquina pára em tempo polinomial $q(|\ f(x)\ |)$ no comprimento da sua entrada, que é $f(x)$. Como f é computável em tempo polinomial, temos $|\ f(x)\ | \leq p(|\ x\ |)$ para alguma polinomial p, pois em cada passo no máximo um símbolo é escrito na saída. Portanto o "passo" (2) leva no máximo tempo $q(p(|\ x\ |))$, que é polinomial.)

Claramente, M reconhece A em tempo polinomial.

(iii) A prova é idêntica ao item (ii), exceto que a máquina de Turing do passo (2) é agora não determinística, e aceita C em tempo polinomial. Com esta modificação obtemos uma máquina não determinística que aceita A em tempo polinomial. ■

Um conjunto $B \subseteq \Sigma^*$ é *$\mathcal{NP}$-m-completo* se

(i) $B \in \mathcal{NP}$,

e

(ii) Para todo $A \in \mathcal{NP}$ temos $A \leq_m^{\mathcal{P}} B$.

COROLÁRIO 2. *Se B é $\mathcal{NP}$-m-completo então $B \in \mathcal{P}$ sse $\mathcal{P} = \mathcal{NP}$.*

Demonstração. Se $\mathcal{P} = \mathcal{NP}$ então $B \in \mathcal{P}$ pois B é *$\mathcal{NP}$-m-completo.*

Reciprocamente, se $B \in \mathcal{P}$, então para todo $A \in \mathcal{NP}$ temos $A \leq_m^{\mathcal{P}} B$. Pela Proposição 1 segue que $A \in \mathcal{P}$. Isto mostra que $\mathcal{NP} \subseteq \mathcal{P}$. Mas pelo Exercício 1, $\mathcal{P} \subseteq \mathcal{NP}$. Portanto $\mathcal{P} = \mathcal{NP}$. ■

O Corolário 2 mostra que se identificarmos algum conjunto B que seja *$\mathcal{NP}$-m-*completo e se acharmos um algoritmo polinomial para B, então podemos reconhecer todos os conjuntos de $\mathcal{NP}$ em tempo polinomial. A classe de conjuntos de $\mathcal{NP}$ é muito ampla e contém um grande número de problemas de interesse prático para os quais não se conhece nenhum algoritmo polinomial. Seria, portanto, um avanço muito grande encontrar um algoritmo polinomial para um dos problemas *$\mathcal{NP}$-m-*completos. Reciprocamente, pela mesma razão parece provável que $\mathcal{P} \neq \mathcal{NP}$. Para provar isto os problemas

$\mathcal{NP}$-*m*-completos são os melhores candidatos para um problema em $\mathcal{NP} \setminus \mathcal{P}$.

Estamos agora em condições de provar o resultado principal deste capítulo.

TEOREMA 4 (Cook). *O conjunto FNC-SAT é $\mathcal{NP}$-m-completo.*

Demonstração. Já vimos no Teorema 2 que *FNC-SAT* $\in \mathcal{NP}$. Precisamos então provar que se $A \in \mathcal{NP}$ então $A \leq_m^{\mathcal{P}}$ *FNC-SAT*. Para isto construiremos, para cada conjunto A aceito em tempo polinomial por uma máquina de Turing não determinística M de fita única, uma função $f_M : \Sigma^*_{e/s} \to \Sigma^*_{e/s}$, computável em tempo polinomial tal que:

(i) $f_M(x)$ é uma fórmula abreviada em forma normal conjuntiva;
(ii) $f_M(x)$ é satisfazível sse $x \in A$, isto é, sse M aceita x.

Estas condições garantem que $A \leq_m^{\mathcal{P}}$ *FNC-SAT* via f_M, e portanto que *FNC-SAT* é $\mathcal{NP}$-*m*-completo.

Seja então $A \in \mathcal{NP}$ e M uma máquina de Turing não determinística de fita única que aceita A em tempo polinomial p. Seja $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_s\}$ o alfabeto de M, onde $\sigma_1 = \vdash$ e $\sigma_2 = \not b$, e $Q = \{q_1, q_2, \ldots, q_e\}$ o conjunto de estados de M, onde q_1 é o estado inicial, q_2 é o estado final q_a, e q_3 o estado final q_r. Seja x uma entrada e seja $T = p(|x|)$. A fórmula $f_M(x)$ terá as seguintes variáveis lógicas com as interpretações padrão indicadas[4]:

VARIÁVEL	INTERPRETAÇÃO PADRÃO
$Estado_t^k$, $1 \leq k \leq e$, $1 \leq t \leq T$;	v sse M no passo t estiver no estado q_k;
$Símbolo\text{-}T_{c,t}^i$, $1 \leq i \leq s$, $1 \leq c, t \leq T$	v sse a célula c da fita de entrada/trabalho/saída contiver o símbolo σ_i no passo t;
$Símbolo\text{-}S_c^i$, $1 \leq i \leq s$, $1 \leq c \leq T$;	v sse a célula c da fita de sugestões contiver o símbolo σ_i;
$Posição\text{-}T_t^c$, $1 \leq c, t \leq T$;	v sse a cabeça da fita de entrada/trabalho/saída estiver sobre a célula c no passo t;
$Posição\text{-}S_t^c$, $1 \leq c, t \leq T$;	v sse a cabeça da fita de sugestões estiver sobre a célula c no passo t.

(4) Na realidade os nomes mnemônicos das variáveis devem ser substituidos por palavras de $\{p, q\}^*$, de acordo com a Seção 4.

Note que o número de variáveis é $eT + sT^2 + sT + T^2 + T^2$ Como $T = p(|\ x\ |)$, e s e e são constantes, temos um número de variáveis polinomial no comprimento da entrada x. Note ainda que, em no máximo T passos, as cabeças de M não podem afastar-se mais que T células para a direita.

A fórmula $f_M(x)$ será construída de tal forma que se M com sugestão y aceitar x em t passos, com $t \leq T$, então $f_M(x)$ com a interpretação padrão torna-se verdadeira. Em outras palavras, se M aceitar x então $f_M(x)$ será satisfazível.

Reciprocamente, se uma interpretação de $f_M(x)$ a tornar verdadeira, então dela podemos extrair uma sugestão y, tal que M com sugestão y aceita x em tempo $t \leq T$.

A fórmula $f_M(x)$ é a conjunção:

$$B \wedge C \wedge D \wedge E \wedge F \wedge G \wedge H \wedge I$$

onde as subfórmulas com a interpretação padrão afirmam:

B – a cabeça da fita de entrada/trabalho/saída em cada passo encontra-se sobre uma e só uma célula;

C – a cabeça da fita de sugestões em cada passo encontra-se sobre uma e só uma célula;

D – cada célula da fita entrada/trabalho/saída em cada passo contém um e um só símbolo;

E – cada célula da fita de sugestões contém um e um só símbolo (note que o conteúdo da fita de sugestões, por ser de leitura apenas, não depende do passo t);

F – em cada passo, M encontra-se em um e um só estado;

G – inicialmente o estado de M é q_1, a fita de entrada/trabalho/saída contém $\vdash x$, a primeira célula da fita de sugestões contém $\vdash$, e as cabeças encontram-se sobre a primeira célula das respectivas fitas;

H – descreve as transições de M;

I – M aceita x em algum passo $t \leq T$.

Cada uma das subfórmulas estará em forma normal conjuntiva abreviada, e assim $f_M(x)$ estará em forma normal conjuntiva abreviada. Descrevemos agora cada uma das subfórmulas. Note que usaremos $\bigwedge_{1 \leq i \leq \ell} A_i$ para denotar a fórmula abreviada $A_1 \wedge A_2 \wedge \ldots \wedge A_\ell$, e $\bigvee_{1 \leq i \leq \ell} A_i$ para denotar a fórmula abreviada $A_1 \vee A_2 \vee \ldots \vee A_\ell$.

$B = \bigwedge_{1 \le t \le T} B_t$, onde

$$B_t = \left(\bigvee_{1 \le c \le T} \textit{Posição-}T_t^c \right) \wedge \bigwedge_{1 \le i < j \le T} (\neg \textit{Posição-}T_t^i \vee \neg \textit{Posição-}T_t^j).$$

Note que $(\neg \textit{Posição-}T_t^i \vee \neg \textit{Posição-}T_t^j)$ afirma que no passo t a cabeça não pode estar simultaneamente sobre as células i e j, onde $i < j$. Assim, B_t afirma que no passo t a cabeça está sobre uma e somente uma das células da fita de entrada/saída/trabalho, e B afirma que isto é verdade em cada passo. As fórmulas C, D, E e F se obtêm analogamente:

$C = \bigwedge_{1 \le t \le T} C_t$, onde

$$C_t = \left(\bigvee_{1 \le c \le T} \textit{Posição-}S_t^c \right) \wedge \bigwedge_{1 \le i < j \le T} (\neg \textit{Posição-}S_t^i \vee \neg \textit{Posição-}S_t^j).$$

$D = \bigwedge_{1 \le c,t \le T} D_{c,t}$ onde

$$D_{c,t} = \left(\bigvee_{1 \le i \le s} \textit{Símbolo-}T_{c,t}^i \right) \wedge$$

$$\bigwedge_{1 \le i < j \le s} (\neg \textit{Símbolo-}T_{c,t}^i \vee \neg \textit{Símbolo-}T_{c,t}^j).$$

$E = \bigwedge_{1 \le c \le T} E_c$, onde

$$E_c = \left(\bigvee_{1 \le i \le s} \textit{Símbolo-}S_c^i \right) \wedge$$

$$\bigwedge_{1 \le i < j \le s} (\neg \textit{Símbolo-}S_c^i \vee \neg \textit{Símbolo-}S_c^j).$$

$F = \bigwedge_{1 \le i \le T} F_t$, onde

$$F_t = \left(\bigvee_{1 \le k \le e} \textit{Estado}_t^k \right) \wedge \bigwedge_{1 \le i < j \le e} (\neg \textit{Estado}_t^i \vee \neg \textit{Estado}_t^j).$$

$G = \textit{Estado}_1^1 \wedge \textit{Posição-}T_1^1 \wedge \textit{Posição-}S_1^1 \wedge$
$\wedge \textit{Símbolo-}S_1^1 \wedge \textit{Símbolo-}T_{1,1}^1 \wedge$
$\wedge \textit{Símbolo-}T_{2,1}^{i_1} \wedge \textit{Símbolo-}T_{3,1}^{i_2} \wedge \ldots \wedge \textit{Símbolo-}T_{n+1,1}^{i_n} \wedge$
$\wedge \textit{Símbolo-}T_{n+2,1}^2 \wedge \textit{Símbolo-}T_{n+3,1}^2 \wedge \ldots \wedge \textit{Símbolo-}T_{T,1}^2$,
onde $x = \sigma_{i_1} \sigma_{i_2} \ldots \sigma_{i_n}$.

$H = H^1 \wedge H^2$, onde

$$H^1 = \bigwedge_{\substack{1 \le i \le s \\ 1 \le c \le T \\ 1 \le t < T}} (\textit{Posição-}T_t^c \vee \neg \textit{Símbolo-}T_{c,t}^i \vee \textit{Símbolo-}T_{c,t+1}^i).$$

(Note que H^1 afirma que, se a cabeça não estiver sobre a célula c da fita de entrada/trabalho/saída no passo t, então o símbolo contido nesta célula não é modificado neste passo.)

$$H^2 = \bigwedge_{\substack{1 \le k \le e \\ 1 \le i,j \le s \\ 1 \le c_1, c_2, t < T}} H_{k,i,j,c_1,c_2,t}, \text{ onde,}$$

supondo que $\delta(q_k, \sigma_i, \sigma_j) = (q_{k'}, \sigma_{i'}, \Delta_1, \Delta_2)$, temos:

$$\begin{aligned}
&H_{k,i,j,c_1,c_2,t} = \\
&(\neg \mathit{Estado}_t^k \vee \neg \mathit{Posição\text{-}T}_t^{c_1} \vee \neg \mathit{Posição\text{-}S}_t^{c_2} \vee \neg \mathit{Símbolo\text{-}T}_{c_1,t}^i \vee \neg \mathit{Símbolo\text{-}S}_{c_2}^j \vee \mathit{Estado}_{t+1}^{k'}) \wedge \\
\wedge\ &(\neg \mathit{Estado}_t^k \vee \neg \mathit{Posição\text{-}T}_t^{c_1} \vee \neg \mathit{Posição\text{-}S}_t^{c_2} \vee \neg \mathit{Símbolo\text{-}T}_{c_1,t}^i \vee \neg \mathit{Símbolo\text{-}S}_{c_2}^j \vee \mathit{Símbolo\text{-}T}_{c_1,t+1}^{i'}) \wedge \\
\wedge\ &(\neg \mathit{Estado}_t^k \vee \neg \mathit{Posição\text{-}T}_t^{c_1} \vee \neg \mathit{Posição\text{-}S}_t^{c_2} \vee \neg \mathit{Símbolo\text{-}T}_{c_1,t}^i \vee \neg \mathit{Símbolo\text{-}S}_{c_2}^j \vee \mathit{Posição\text{-}T}_{t+1}^{c_1+\Delta_1}) \wedge \\
\wedge\ &(\neg \mathit{Estado}_t^k \vee \neg \mathit{Posição\text{-}T}_t^{c_1} \vee \neg \mathit{Posição\text{-}S}_t^{c_2} \vee \neg \mathit{Símbolo\text{-}T}_{c_1,t}^i \vee \neg \mathit{Símbolo\text{-}S}_{c_2}^j \vee \mathit{Posição\text{-}S}_{t+1}^{c_2+\Delta_2})
\end{aligned}$$

(Assim, H^2 afirma que para todo passo $t < T$, se M estiver no estado q_k, a cabeça da fita de entrada/saída/trabalho estiver sobre a célula c_1 que contiver o símbolo σ_i, e a cabeça da fita de sugestões estiver sobre a célula c_2 que contiver o símbolo σ_j, então no passo $t+1$ a máquina M estará no estado $q_{k'}$, a célula c_1 da fita de entrada/saída/trabalho conterá $\sigma_{i'}$, e as cabeças estarão respectivamente sobre as células $c_1 + \Delta_1$ e $c_2 + \Delta_2$).

Finalmente

$$I = \bigvee_{1 \le t \le T} \mathit{Estado}_t^2.$$

Claramente, se M aceitar x com alguma sugestão $y = \sigma_{j_1} \sigma_{j_2} \ldots \sigma_{j_m}$, então a interpretação padrão das variáveis torna $f_M(x)$ verdadeira, e assim, a fórmula $f_M(x)$ é neste caso satisfazível. Reciprocamente, se $f_M(x)$ for satisfazível para alguma interpretação das variáveis então sejam

$$\mathit{Símbolo\text{-}S}_1^{j_1},\ \mathit{Símbolo\text{-}S}_2^{j_2},\ \ldots,\ \mathit{Símbolo\text{-}S}_T^{j_T}$$

as variáveis $\mathit{Símbolo\text{-}S}_c^j$ verdadeiras nesta interpretação. (Como a interpretação torna E verdadeira, para cada c existe um e um só j tal que $\mathit{Símbolo\text{-}S}_c^j$ é verdadeira.) Conseqüentemente M aceita x com sugestão $y = \sigma_{j_1} \sigma_{j_2} \ldots \sigma_{j_T}$, pois as variáveis de $f_M(x)$ que são verdadeiras nesta interpretação descrevem, quando reinterpretadas pela interpretação padrão, a computação de M que aceita x com esta sugestão. Assim, $f_M(x)$ é satisfazível sse M aceita x com alguma sugestão y, isto é

$$f_M(x) \in \mathit{FNC\text{-}SAT} \text{ sse } x \in A.$$

Como o comprimento de cada subfórmula de $f_M(x)$ é polinomial em $|x|$, temos que $|f_M(x)|$ é polinomial em $|x|$. Seguindo a descrição acima de $f_M(x)$, podemos construir uma máquina de Turing que com entrada x escreve a fórmula $f_M(x)$ em tempo polinomial. Assim, $f_M(x)$ é computável em tempo polinomial, o que completa a demonstração. ■

COROLÁRIO 3. *O conjunto SAT é $\mathcal{NP}$-m-completo.*

Demonstração. A função $f_M(x)$ do Teorema 4 está em forma normal conjuntiva abreviada para qualquer x. Portanto, como $A \leq_m^{\mathcal{P}} FNC\text{-}SAT$ via $f_M(x)$, é imediato que $f_M(x)$ também m-reduz A a SAT em tempo polinomial. Pelo Exercício 8, $SAT \in \mathcal{NP}$. Portanto SAT é $\mathcal{NP}$-m-completo. ■

6. Outros problemas NP-m-completos

Após a descoberta de Cook de que $FNC\text{-}SAT$ e SAT são $\mathcal{NP}$-m-completos, constatou-se que uma lista extensa de problemas de importância prática, lista que continua crescendo a cada ano, são também $\mathcal{NP}$-m-completos. Note que uma vez demonstrado que um conjunto A é $\mathcal{NP}$-m-completo, basta mostrar que $A \leq_m^{\mathcal{P}} B$ e $B \in \mathcal{NP}$, para demonstrar que B também é $\mathcal{NP}$-m-completo. Estas reduções, em geral, são bem mais simples do que a redução que vimos no Teorema 4. Karp [57] exibiu uma lista de 21 problemas $\mathcal{NP}$-m-completos dos quais extraímos seis no teorema abaixo.

Percebeu-se, então, que as soluções rápidas para um grande número de problemas de interesse estão interrelacionadas no sentido de que ou cada um deles ou nenhum deles pode ser resolvido rapidamente por um algoritmo. Isto explica a importância prática da questão $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$.

Também do ponto de vista teórico, vê-se que esta é uma questão central a ser respondida para se ter um melhor entendimento de algoritmos rápidos. Com efeito, devido ao Teorema 4, responder à questão $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$ resultará num avanço significativo nesta área, qualquer que seja a resposta: se $\mathcal{P} = \mathcal{NP}$ então teremos algoritmos polinomiais para todos os problemas em $\mathcal{NP}$, que, como se viu, inclui um grande número de problemas para os quais não dispomos de algoritmos polinomiais no momento; se, por outro lado $\mathcal{P} \neq \mathcal{NP}$, então sabe-

remos que um grande número de problemas, os problemas $\mathcal{NP}$-*m*-completos, não podem ser resolvidos por algoritmos polinomiais.

Damos abaixo, sem demonstração, alguns dos problemas $\mathcal{NP}$-*m*-completos descobertos por Karp.

TEOREMA 5. *Os problemas seguintes são* $\mathcal{NP}$-*m-completos*:

(i) *Programação inteira* $0 - 1$:
$\{(C, \mathbf{d}) \mid C$ *é uma matriz inteira,* $\mathbf{d}$ *é um vetor, e existe um vetor* $\mathbf{x}$ *sobre* $\{0, 1\}$, *tal que* $C\mathbf{x} \geq \mathbf{d}\}$;

(ii) *Classificação ótima (veja as definições necessárias no Capítulo C. VI)*:
$\{(D, k) \mid o$ *grafo orientado* D *admite uma classificação* f, *tal que no máximo* k *arestas de* D *são inversões com relação a* $f\}$;

(iii) *Circuito hamiltoniano (veja as definições necessárias no Capítulo C. I)*:
$\{G \mid o$ *grafo* G *tem um circuito que passa por todos os vértices de* $G\}$;

(iv) *Número cromático (veja as definições necessárias no Capítulo C.IV)*:
$\{(G, k) \mid o$ *grafo* G *é* k-*colorável*$\}$;

(v) *Emparelhamento em três dimensões (confronte com o Capítulo C.III)*:
$\{(T, U) \mid U \subseteq T \times T \times T, T$ *é um conjunto finito, e existe um subconjunto* W *de* U, *tal que* $|W| = |T|$ *e para cada* $i = 1, 2, 3$ $\bigcup_{(w_1, w_2, w_3) \in W} \{w_i\} = T\}$;

(vi) *Problema da mochila*:
$\{(n, a_1, a_2, \ldots, a_n, b) \mid n, a_1, \ldots, a_n, b \in \mathbb{N}, n \geq 1,$ *e existem* $x_1, x_2, \ldots, x_n \in \{0, 1\}$, *tais que* $\sum_{1 \leq i \leq n} a_i x_i = b\}$.

Como vimos, não se sabe se $\mathcal{P} = \mathcal{NP}$ ou $\mathcal{P} \neq \mathcal{NP}$. A hipótese mais popular, embora não unânime, entre os cientistas de computação, parece ser $\mathcal{P} \neq \mathcal{NP}$, que aparenta ser mais plausível tendo em vista o grande número de problemas de interesse em $\mathcal{NP}$ para os quais não se conhecem algoritmos polinomiais. Não se sabe, tampouco, se $\mathcal{NP}$ é fechado sob complementação. Como $\mathcal{P}$ é obviamente fechado sob complementação é claro que se $\mathcal{NP}$ não for fechado sob complementação então $\mathcal{P} \neq \mathcal{NP}$. Note que $\mathcal{NP}$ é fechado sob complementação sse o complemento de algum conjunto $\mathcal{NP}$-*m*-completo pertencer a $\mathcal{NP}$. Outra vez, a hipótese mais popular, e pelas mesmas razões, é que $\mathcal{NP}$ não é fechado sob complementação.

Na Seção 2 descrevemos uma máquina de Turing não determinística que aceita o conjunto *COMPOSTO* de todas as palavras que

são a representação decimal de um número composto. Esta máquina é polinomial, e portanto *COMPOSTO* pertence a $\mathcal{NP}$. Menos evidente é o fato de que o complemento de *COMPOSTO*, o conjunto *PRIMO* das palavras que representam números primos na notação decimal, também pertence a $\mathcal{NP}$. Não se sabe se *PRIMO* pertence a $\mathcal{P}$, embora Miller tenha provado que se a hipótese estendida de Riemann for verdadeira então *PRIMO* pertence a $\mathcal{P}$. Não se sabe, tampouco, se *PRIMO* é $\mathcal{NP}$-*m*-completo, mas parece provável que não seja, pois se fosse então $\mathcal{NP}$ seria fechado sob complementação.

EXERCÍCIOS

1. Verifique que $\mathcal{P}$ é um subconjunto de $\mathcal{NP}$.
2. Verifique que os pares de fórmulas abaixo são logicamente equivalentes
   (i) $(p \vee \neg p)$ e v;
   (ii) $(p \wedge \neg p)$ e f;
   (iii) $(p \wedge (pp \wedge pq))$ e $((p \wedge pp) \wedge pq)$ (propriedade associativa de $\wedge$);
   (iv) $(p \vee (pp \vee pq))$ e $((p \vee pp) \vee pq)$ (propriedade associativa de $\vee$);
   (v) $(p \wedge q)$ e $(q \wedge p)$ (propriedade comutativa de $\wedge$);
   (vi) $(p \vee q)$ e $(q \vee p)$ (propriedade comutativa de $\vee$);
   (vii) $(p \wedge (pp \vee pq))$ e $((p \wedge pp) \vee (p \wedge pq))$ (propriedade distributiva de $\wedge$ sobre $\vee$);
   (viii) $(p \vee (pp \wedge pq))$ e $((p \vee pp) \wedge (p \vee pq))$ (propriedade distributiva de $\vee$ sobre $\wedge$);
   (ix) $\neg(p \wedge q)$ e $(\neg p \vee \neg q)$ (lei de DeMorgan);
   (x) $\neg(p \vee q)$ e $(\neg p \wedge \neg q)$ (lei de DeMorgan);
   (xi) p e $\neg\neg p$;
   (xii) v e $\neg f$;
   (xiii) f e $\neg v$.
3. Mostre que para toda fórmula A existem fórmulas B e C, logicamente equivalentes a A, respectivamente em forma normal disjuntiva e conjuntiva. Note que B e C podem ser fórmulas exponencialmente mais compridas do que A.
4. Mostre que *FND-SAT* e *FNC-TAUT* pertencem a $\mathcal{P}$.
5. Seja $\equiv_m^{\mathcal{P}}$ a relação definida por $A \equiv_m^{\mathcal{P}} B$ sse $A \leq_m^{\mathcal{P}} B$ e $B \leq_m^{\mathcal{P}} A$. Mostre que $\equiv_m^{\mathcal{P}}$ é uma relação de equivalência.

6. Seja $p(x)$ um polinômio com coeficientes inteiros, e seja $\leq_m^{p(n)}$ a relação definida por: $A \leq_m^{p(n)} B$ sse existir uma função $f: \Sigma^* \to \Sigma^*$ computável em tempo $p(n)$ tal que $x \in A$ sse $f(x) \in B$. Prove que $\leq_m^{p(n)}$ não é transitiva.
7. Prove que existe uma máquina de Turing determinística que tendo por entrada uma fórmula em forma normal conjuntiva dá por saída a fórmula abreviada correspondente em tempo polinomial.
8. Prove que *SAT* pertence a $\mathcal{NP}$.
9. Como *SAT* pertence a $\mathcal{NP}$ pelo exercício anterior, o Teorema 4 garante que dada uma fórmula x existe uma fórmula z de comprimento polinomial em $|x|$ e em forma normal conjuntiva, tal que z é satisfazível sse x é satisfazível. Exiba uma tal fórmula explicitamente. Confronte com o Exercício 3.
10. Mostre que cada um dos problemas do Teorema 5 está em $\mathcal{NP}$.
11. Seja *k-FNC-SAT* o subconjunto de *FNC-SAT* tal que cada fórmula de *k-FNC-SAT* é uma conjunção de disjunções de no máximo k átomos. Mostre que
    (i) *k-FNC-SAT* pertence a $\mathcal{NP}$.
    (ii) Se $k \geq 3$ *k-FNC-SAT* é $\mathcal{NP}$-*m*-completo.
    (iii) 2-*FNC-SAT* pertence a $\mathcal{P}$.
12. Sejam $H_1 = \{G \mid G$ é um grafo orientado que tem um circuito orientado que passa por todos as vértices de $G\}$, e $H_2 = \{G \mid G$ é um grafo que tem um circuito que passa por todos os vértices de $G\}$. Prove que:
    (i) $H_2 \leq_m^{\mathcal{P}} H_1$;
    (ii) $H_1 \leq_m^{\mathcal{P}} H_2$.
13. Construa um grafo orientado G com três vértices *iniciais* v_1, v_2 e v_3, e três correspondentes vértices *finais* v_1', v_2' e v_3' tal que:
    (i) Dados quaisquer i vértices iniciais ($i = 1, 2$ ou 3), existem i caminhos orientados disjuntos nos vértices, com origens respectivamente nos vértices dados e términos respectivamente nos vértices finais correspondentes, caminhos tais que em cada um dos vértices de G passa exatamente um dos i caminhos. (A união disjunta dos vértices dos i caminhos é, portanto, VG.)
    (ii) Para cada $i = 1, 2$ ou 3, *não* existem i caminhos orientados disjuntos nos vértices, com origens em vértices iniciais e

términos em vértices finais, e tais que em cada vértice de G passe exatamente um dos i caminhos, *exceto* se para cada um dos i caminhos o seu término for o vértice final correspondente à sua origem.
(Veja as definições necessárias nos Capítulos C.I e C.VI.)

14. Mostre que $3\text{-}FNC\text{-}SAT \leq_m^{\mathscr{P}} H_1$ (vide Exercícios 11 e 12). Note que isto demonstra, com os Exercícios 10, 11, e 12, o item (iii) do Teorema 5. (Sugestão: Use um grafo orientado como o construído no Exercício 13 para representar cada disjunção da fórmula.)
15. Demonstre o Teorema 5.

NOTAS BIBLIOGRÁFICAS

A classe $\mathscr{P}$ foi primeiro definida em [16] por Cobham. O artigo de Edmonds [23] contém uma discussão informal de o que deve ser considerado um algoritmo eficiente e sugere também nas entrelinhas a classe $\mathscr{P}$. Cook provou em [17] os resultados principais deste capítulo, e caracterizou pela prima vez a questão $\mathscr{P} \stackrel{?}{=} \mathscr{NP}$ precisamente. Karp [57] exibiu uma lista de 21 problemas $\mathscr{NP}$-m-completos, demonstrando o impacto da questão levantada por Cook. Meyer e Stockmeyer, [80] e [115], provaram os primeiros resultados estabelecendo limites inferiores superpolinomiais, e mesmo superexponenciais para vários problemas naturais. O resultado que $PRIMO \in \mathscr{NP}$ encontra-se em [96], e [81] contém o algoritmo mais rápido conhecido para reconhecer *PRIMO*. O melhor limite superior conhecido para este algoritmo é exponencial, mas a hipótese estendida de Riemann implicaria que o algoritmo é polinomial [81].