CAPÍTULO III

PRODUTO EFICIENTE DE MATRIZES

1. Introdução

Quantas multiplicações são necessárias para obter o produto de duas matrizes?

Para responder a esta pergunta devemos precisar os métodos admissíveis para o cálculo do produto. Por exemplo, se os elementos das matrizes forem números reais positivos, e se permitirmos as operações *logaritmo* e *exponencial*, então nenhuma multiplicação é necessária, já que

$$a \cdot b = \textit{exponencial}\,(\textit{logaritmo}\,(a) + \textit{logaritmo}\,(b)).$$

Utilizando a identidade acima podemos eliminar todas as multiplicações para achar o produto de duas matrizes de números reais e positivos.

Suporemos então, que os elementos das matrizes a serem multiplicadas pertencem a um anel com unidade A. Assim, as únicas operações permitidas são as operações do anel, além de testes de igualdade e desigualdade. Note que não supusemos que o anel é comutativo, isto é, $a \cdot b$ pode não ser igual a $b \cdot a$[1]. A nossa pergunta, portanto, fica: dadas duas matrizes $n \times n$ X e Y sobre A, quantas operações de A serão necessárias para calcular o produto $X \cdot Y$?

---

(1) Lembramos que as operações $+$ e $\cdot$ gozam das propriedades:

1. $(A, +)$ é um *grupo abeliano*, isto é,
   (a) $a + (b + c) = (a + b) + c$ para todo $a, b, c \in A$;
   (b) $a + b = b + a$ para todo $a, b \in A$;
   (c) Existe um elemento $0 \in A$ tal que $a + 0 = a$ para todo $a \in A$;
   (d) Para cada $a \in A$, existe um elemento $(-a) \in A$, chamado de o *simétrico* de a, tal que $a + (-a) = 0$;
2. $(A, \cdot)$ é um *monóide*, isto é,
   (e) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ para todo $a, b, c \in A$;
   (f) Existe um elemento $1 \in A$, $1 \neq 0$, tal que $a \cdot 1 = 1 \cdot a = a$ para todo $a \in A$;
3. A operação $\cdot$ *distribui* sobre $+$, isto é, para todo a, b, $c \in A$
   (g) $((a + b) \cdot c) = (a \cdot c) + (b \cdot c)$;
   (h) $(c \cdot (a + b)) = (c \cdot a) + (c \cdot b)$.

Esta pergunta, como vimos no Capítulo I, é típica dos problemas de complexidade de computação, no sentido de pedir a determinação da quantidade de recursos computacionais, (operações • no caso), necessárias para executar uma certa tarefa. O algoritmo clássico de produto de matrizes $n \times n$ nos fornece um limite superior de n^3 operações. Com efeito, se

$$Z = [z_{ij}] = X \cdot Y, \text{ onde } X = [x_{ij}] \text{ e } Y = [y_{ij}]$$

então

$$z_{ij} = \sum_{k=1}^{n} x_{ik} \cdot y_{kj}, \quad 1 \leq i, j \leq n,$$

onde Σ denota a somatória no anel A. Portanto, para encontrar z_{ij} utilizamos uma multiplicação para cada valor de k, e assim, são utilizadas n multiplicações neste cálculo. Como Z possui n^2 elementos, utilizamos $n \cdot n^2 = n^3$ multiplicações no cálculo de Z. O número de somas, pelo mesmo raciocínio, será $n - 1$ para o cálculo de cada z_{ij}, e portanto,

$$n^2 \cdot (n - 1) = n^3 - n^2$$

para a matriz Z toda.

Intuitivamente, parece à primeira vista que o algoritmo acima é ótimo, e assim nada nos restaria fazer para melhorá-lo, exceto demonstrar que de fato são necessárias n^3 multiplicações para determinar Z. Como veremos entretanto, nossa intuição nos engana neste caso, pois, surpreendentemente, o algoritmo clássico pode ser consideravelmente melhorado.

2. O algoritmo de Strassen

Para achar o produto de duas matrizes 1×1, certamente precisaremos de uma multiplicação (vide Exercício 8). Para matrizes 1×1, portanto, n^3 multiplicações são necessárias e suficientes. Já para matrizes 2×2 não se consegiu provar o limite inferior $n^3 = 2^3 = 8$ multiplicações. Isto não é de se admirar, pois em 1969 Strassen observou que é possível multiplicar matrizes 2×2 usando apenas 7 mul-

tiplicações, com o algoritmo abaixo (expresso na linguagem *LP*, definida na Parte *A*, onde $+$, $-$ e $\cdot$ denotam as respectivas operações no anel *A*):

procedimento *Prodmat2*(X, Y):
início
$p_1 \leftarrow (x_{11} + x_{22}) \cdot (y_{11} + y_{22})$;
$p_2 \leftarrow (x_{21} + x_{22}) \cdot y_{11}$;
$p_3 \leftarrow x_{11} \cdot (y_{12} - y_{22})$;
$p_4 \leftarrow x_{22} \cdot (y_{21} - y_{11})$;
$p_5 \leftarrow (x_{11} + x_{12}) \cdot y_{22}$;
$p_6 \leftarrow (x_{21} - x_{11}) \cdot (y_{11} + y_{12})$;
$p_7 \leftarrow (x_{12} - x_{22}) \cdot (y_{21} + y_{22})$;
$z_{11} \leftarrow p_1 + p_4 - p_5 + p_7$;
$z_{12} \leftarrow p_3 + p_5$;
$z_{21} \leftarrow p_2 + p_4$;
$z_{22} \leftarrow p_1 + p_3 - p_2 + p_6$;
devolva Z $\{Z = [z_{ij}]\}$
fim

Com as técnicas recursivas que foram vistas na Parte *A*, podemos utilizar o algoritmo *Prodmat2* para obter um método mais eficiente que o algoritmo clássico, para o produto de duas matrizes quadradas $n \times n$ X e Y. Inicialmente vamos supor $n = 2^k$, para algum $k \geq 2$. Mais tarde mostraremos como eliminar esta restrição. Se $n = 2^k$, podemos subdividir as matrizes X e Y, cortando-as em quatro partes de mesma dimensão:

$$X = \left[\begin{array}{c|c} X_{11} & X_{12} \\ \hline X_{21} & X_{22} \end{array}\right] \quad \text{e} \quad Y = \left[\begin{array}{c|c} Y_{12} & Y_{12} \\ \hline Y_{21} & Y_{22} \end{array}\right].$$

Assim X_{ij} e Y_{ij} são por sua vez matrizes quadradas $n/2 \times n/2$ chamadas *blocos*. Por exemplo, X_{11} é a matriz formada pelas $n/2$ primeiras linhas e $n/2$ primeiras colunas de X, enquanto que X_{12} é a matriz formadas pelas $n/2$ primeiras linhas e $n/2$ últimas colunas de X, e assim por diante. É um fato bem conhecido da Álgebra Linear que o produto de matrizes pode ser feito por blocos, isto é

$$Z = X \cdot Y = \left[\begin{array}{c|c} Z_{11} & Z_{12} \\ \hline Z_{21} & Z_{22} \end{array}\right], \text{ onde } Z_{ij} = X_{i1} \cdot Y_{1j} + X_{i2} \cdot Y_{2j}$$

e as multiplicações e somas indicadas são agora sobre o anel das matrizes $n/2 \times n/2$. Em outras palavras, podemos efetuar o produto $X \cdot Y$ como se X e Y fossem matrizes 2×2 com elementos tomados do anel das matrizes $n/2 \times n/2$, que reinterpretado como uma matriz $n \times n$ sobre o anel A, coincide com o produto de matrizes $n \times n$ $X \cdot Y$. Por exemplo, para $n = 4$ considere

$$X = \left[\begin{array}{cc|cc} 1 & 0 & 2 & 1 \\ 2 & 1 & -1 & 0 \\ \hline 2 & 0 & 0 & 1 \\ 0 & 3 & 1 & 0 \end{array}\right] \text{ e } Y = \left[\begin{array}{cc|cc} 0 & 1 & 2 & 0 \\ 1 & 2 & -1 & 2 \\ \hline 1 & 1 & 0 & 0 \\ 2 & 0 & 0 & -2 \end{array}\right]$$

Portanto,

$$Z = X \cdot Y = \left[\begin{array}{cc|cc} 4 & 3 & 2 & -2 \\ 0 & 3 & 3 & 2 \\ \hline 2 & 2 & 4 & -2 \\ 4 & 7 & -3 & 6 \end{array}\right].$$

Por outro lado,

$$X_{11} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}; X_{12} = \begin{bmatrix} 2 & 1 \\ -1 & 0 \end{bmatrix}; Y_{11} = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}; Y_{12} = \begin{bmatrix} 2 & 0 \\ -1 & 2 \end{bmatrix}$$

$$X_{21} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}; X_{22} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; Y_{21} = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}; Y_{22} = \begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix}$$

Assim,

$$Z_{11} = X_{11} \cdot Y_{11} + X_{12} \cdot Y_{21} = \begin{bmatrix} 0 & 1 \\ 1 & 4 \end{bmatrix} + \begin{bmatrix} 4 & 2 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} 4 & 3 \\ 0 & 3 \end{bmatrix};$$

$$Z_{12} = X_{11} \cdot Y_{12} + X_{12} \cdot Y_{22} = \begin{bmatrix} 2 & 0 \\ 3 & 2 \end{bmatrix} + \begin{bmatrix} 0 & -2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & -2 \\ 3 & 2 \end{bmatrix};$$

$$Z_{21} = X_{21} \cdot Y_{11} + X_{22} \cdot Y_{21} = \begin{bmatrix} 0 & 2 \\ 3 & 6 \end{bmatrix} + \begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \\ 4 & 7 \end{bmatrix};$$

$$Z_{22} = X_{21} \cdot Y_{12} + X_{22} \cdot Y_{22} = \begin{bmatrix} 4 & 0 \\ -3 & 6 \end{bmatrix} + \begin{bmatrix} 0 & -2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 4 & -2 \\ -3 & 6 \end{bmatrix}.$$

A idéia do algoritmo para calcular o produto de X e Y é subdividir as matrizes em blocos $n/2 \times n/2$, e em seguida aplicar a rotina *Prodmat*2, com X e Y consideradas como matrizes 2×2 sobre o anel das matrizes $n/2 \times n/2$. Os produtos e somas de *Prodmat*2 são portanto produtos e somas de matrizes $n/2 \times n/2$. Para efetuar cada um destes 7 produtos de matrizes, chamamos o algoritmo recursivamente, agora com argumentos menores $n/2 \times n/2$. Damos abaixo a descrição precisa do algoritmo em *LP*. A notação $X[i : j, k : \ell]$ significará o bloco de X formado pelas linhas de i a j, e as colunas de k a ℓ, inclusive.

procedimento *Prodmat*(X, Y, n): {Produto de matrizes $n \times n$, $n = 2^k$}
início

se $n = 2$ **então devolva** *Prodmat*2(X, Y) **senão** $m \leftarrow n/2$;
$X_{11} \leftarrow X[1 : m, 1 : m]$; $Y_{11} \leftarrow Y[1 : m, 1 : m]$;
$X_{12} \leftarrow X[1 : m, m + 1 : n]$; $Y_{12} \leftarrow Y[1 : m, m + 1 : n]$;
$X_{21} \leftarrow X[m + 1 : n, 1 : m]$; $Y_{21} \leftarrow Y[m + 1 : n, 1 : m]$;
$X_{22} \leftarrow X[m + 1 : n, m + 1 : n]$; $Y_{22} \leftarrow Y[m + 1 : n, m + 1 : n]$;
$P_1 \leftarrow$ *Prodmat*$(S(X_{11}, X_{22}, m, \text{“+”}), S(Y_{11}, Y_{22}, m, \text{“+”}), m)$;
$\{P_1 = (X_{11} + X_{22}) \cdot (Y_{11} + Y_{22})\}$

$P_2 \leftarrow Prodmat(S(X_{21}, X_{22}, m, \text{“+”}), Y_{11}, m);$
$\{P_2 = (X_{21} + X_{22}) \cdot Y_{11}\}$
$P_3 \leftarrow Prodmat(X_{11}, S(Y_{12}, Y_{22}, m, \text{“−”}), m);$
$\{P_3 = X_{11} \cdot (Y_{12} - Y_{22})\}$
$P_4 \leftarrow Prodmat(X_{22}, S(Y_{21}, Y_{11}, m, \text{“−”}), m);$
$\{P_4 = X_{22} \cdot (Y_{21} - Y_{11})\}$
$P_5 \leftarrow Prodmat(S(X_{11}, X_{12}, m, \text{“+”}), Y_{22}, m);$
$\{P_5 = (X_{11} + X_{12}) \cdot Y_{22}\}$
$P_6 \leftarrow Prodmat(S(X_{21}, X_{11}, m, \text{“−”}), S(Y_{11}, Y_{12}, m, \text{“+”}), m);$
$\{P_6 = (X_{21} - X_{11}) \cdot (Y_{11} + Y_{12})\}$
$P_7 \leftarrow Prodmat(S(X_{12}, X_{22}, m, \text{“−”}), S(Y_{21}, Y_{22}, m, \text{“+”}), m);$
$\{P_7 = (X_{12} - X_{22}) \cdot (Y_{21} + Y_{22})\}$
$Z_{11} \leftarrow S(S(S(P_1, P_4, m, \text{“+”}), P_5, m, \text{“−”}), P_7, m, \text{“+”});$
$\{Z_{11} = P_1 + P_4 - P_5 + P_7\}$
$Z_{12} \leftarrow S(P_3, P_5, m, \text{“+”});$
$\{Z_{12} = P_3 + P_5\}$
$Z_{21} \leftarrow S(P_2, P_4, m, \text{“+”});$
$\{Z_{21} = P_2 + P_4\}$
$Z_{22} \leftarrow S(S(S(P_1, P_3, m, \text{“+”}), P_2, m, \text{“−”}), P_6, m, \text{“+”});$
$\{Z_{22} = P_1 + P_3 - P_2 + P_6\}$
$Z[1 : m, 1 : m] \leftarrow Z_{11}$; $Z[1 : m, m + 1 : n] \leftarrow Z_{12}$;
$Z[m + 1 : n, 1 : m] \leftarrow Z_{21}$; $Z[m + 1 : n, m + 1 : n] \leftarrow Z_{22}$;
devolva Z $\{Z = X \cdot Y\}$

fim

procedimento $S(A, B, m, s)$: $\{A\ s\ B$, onde $s =$ “+” ou $s =$ “−”$\}$

início
$i \leftarrow 1$;
enquanto $i \leq m$ **faça**
início
$j \leftarrow 1$;
enquanto $j \leq m$ **faça**
início
se $s =$ “+” **então** $c_{ij} \leftarrow a_{ij} + b_{ij}$
senão $c_{ij} \leftarrow a_{ij} - b_{ij}$;
$j \leftarrow j + 1$
fim;
$i \leftarrow i + 1$
fim;
devolva C $\{C = A\ s\ B\}$
fim

Vamos agora calcular o número de multiplicações $M(n)$ utilizadas por *Prodmat* para calcular o produto de duas matrizes $n \times n$, onde $n = 2^k$. Se $k = 1$ então o número de multiplicações é 7 pois neste caso chamamos diretamente *Prodmat*2. Para $k \geq 2$ chamamos *Prodmat* recursivamente, para multiplicar matrizes $n/2 \times n/2$, sete vezes. O número de multiplicações utilizadas em cada uma destas chamadas recursivas é $M(2^{k-1})$. Portanto, temos

$$\begin{cases} M(2^1) = 7 \\ M(2^k) = 7 \cdot M(2^{k-1}) \text{ para } k > 1. \end{cases}$$

A solução desta recorrência é $M(2^k) = 7^k$ (vide o Exercício 3). Lembrando que $n = 2^k$, temos:

$$M(n) = 7^k = 7^{\log_2 n} = n^{\log_2 7}.$$

Conseguimos, portanto, reduzir o número de multiplicações de n^3 para $n^{\log_2 7}$, que é uma economia considerável, já que $\log_2 7 = 2{,}80735\ldots$.

O leitor atento terá observado que *Prodmat*2 economiza uma multiplicação à custa de quatorze somas. Como *Prodmat* é baseado em *Prodmat*2, pareceria, à primeira vista, que a economia que obtivemos no número de multiplicações poderia ser anulada por um aumento no número de somas. Um exame mais minucioso da situação revela, no entanto, que *Prodmat* na realidade utiliza menos operações de soma do que o algoritmo clássico, exceto para valores pequenos de n. Com efeito seja $A(n)$ o número de adições que *Prodmat* utiliza para $n = 2^k$. Se $n = 2$ então este número é 18, o número de somas (ou subtrações) que *Prodmat*2 utiliza. Para $k > 1$ chamamos *Prodmat* recursivamente 7 vezes com matrizes $n/2 \times n/2$. Cada uma destas chamadas utiliza portanto $A(2^{k-1})$ somas. Antes de chamar *Prodmat*, no entanto, somas são utilizadas também na rotina de soma de matrizes S, que usa $(n/2)^2$ somas[2]. A rotina S é chamada 18 vezes em *Prodmat*.

(2) Contamos aqui apenas as somas no anel A. Não estamos contando as somas necessárias para o controle do processo, nos incrementos às variáveis i e j.

Portanto, temos

$$\begin{cases} A(2^1) = 18 \\ A(2^k) = 7 \cdot A(2^{k-1}) + 18 \cdot (2^{k-1})^2. \end{cases}$$

A solução desta recorrência é $A(2^k) = 6 \cdot (7^k - 4^k)$ (vide o Exercício 3). O número de somas utilizadas por *Prodmat* é, portanto, menor do que $6 \cdot 7^k$, isto é, menor do que $6 \cdot n^{\log_2 7} < n^3 - n^2$ para n suficientemente grande.

Vejamos agora o que fazer se n não for uma potência de 2. Seja $2^{k-1} < n < 2^k$. Essencialmente o algoritmo que usaremos é o *Prodmat* visto acima, chamado com a potência de 2 mais próxima. Para isto preenchemos com zeros as $2^k - n$ linhas e colunas extras criadas, conforme o esquema abaixo.

$$X = \begin{array}{c} \overbrace{}^{n} \\ \left.\begin{array}{|c|c|} \hline X & 0\ 0\ \ldots\ 0 \\ & 0\ 0\ \ldots\ 0 \\ & \ldots\ldots \\ \hline 0\ 0\ \ldots & \ldots\ldots\ 0 \\ \ldots\ldots & \ldots\ldots\ldots \\ 0\ 0\ \ldots & \ldots\ldots\ 0 \\ \hline \end{array}\right\} n \\ \underbrace{}_{2^k} \end{array}; \qquad Y' = \left[\begin{array}{|c|c|} \hline Y & 0\ 0\ \ldots\ 0 \\ & 0\ 0\ \ldots\ 0 \\ & \ldots\ldots \\ \hline 0\ 0\ \ldots & \ldots\ldots\ 0 \\ \ldots\ldots & \ldots\ldots\ldots \\ 0\ 0\ \ldots & \ldots\ldots\ 0 \\ \hline \end{array}\right];$$

$$Z' = X' \cdot Y' = \left[\begin{array}{|c|c|} \hline X \cdot Y & 0\ 0\ \ldots\ 0 \\ & 0\ 0\ \ldots\ 0 \\ & \ldots\ldots \\ \hline 0\ 0\ \ldots & \ldots\ldots\ 0 \\ \ldots\ldots & \ldots\ldots\ldots \\ 0\ 0\ \ldots & \ldots\ldots\ 0 \\ \hline \end{array}\right].$$

É fácil ver que o número de multiplicações será menor que $7n^{\log_2 7}$ e o número de somas menor que $42n^{\log_2 7}$.

Vamos entender melhor o mecanismo pelo qual Strassen conseguiu reduzir, assintoticamente, isto é, para n suficientemente grande, tanto o número de multiplicações como o número de somas. Inicialmente, vamos admitir que é o número de multiplicações que queremos

realmente minimizar, isto é, admitamos que o custo de uma multiplicação no anel seja muito maior que o custo de uma soma. A idéia de Strassen é então a seguinte: vamos minimizar o número de multiplicações num caso particular ($n = 2$), sem nos preocupar com o número de somas, já que o custo de cada multiplicação é muito maior que o custo de uma soma. Se conseguirmos um algoritmo melhor para o caso particular, que se aplique em qualquer anel não comutativo, então no caso geral podemos tirar proveito desta economia pelo artifício de dividir as matrizes de entrada originais em blocos, de tal maneira que possamos aplicar o algoritmo particular às matrizes de entrada, consideradas como matrizes de blocos. (A hipótese da não comutatividade do anel é aqui usada, já que o produto de matrizes é em geral não comutativo.) Desta forma reduzimos o problema original a produtos e somas de matrizes menores ($n/2 \times n/2$, no nosso caso). A economia no número de multiplicações no caso particular, se transformará aqui num número menor de produtos de matrizes do que no algoritmo clássico, e poderemos aumentar ainda mais esta economia se reaplicarmos recursivamente o método para calcular estes produtos de matrizes. Por outro lado, o aumento no número de somas no algoritmo particular se transforma num aumento no número de somas de matrizes ($n/2 \times n/2$, no nosso caso). Mas somar matrizes é mais fácil do que multiplicar matrizes, exigindo, pelo método clássico, da ordem de n^2 somas contra da ordem de n^3 multiplicações e somas para cada produto de matrizes, e portanto é vantajoso trocar um número menor de produtos de matrizes por um número maior de somas de matrizes, ainda que a hipótese original, de que multiplicações são muito mais caras do que somas no nosso anel, fosse falsa. É devido a este fato que, surpreendentemente, o algoritmo acaba economizando até no número de somas para n suficientemente grande.

Como vimos, o algoritmo de Strassen economiza no número de multiplicações utilizadas para calcular o produto de duas matrizes, mas para n pequeno aumenta o número de somas. Se o custo de cada multiplicação for muito maior que o custo de cada soma então o algoritmo poderá ser aplicado com vantagem mesmo para n relativamente pequeno. Se isto não acontecer, então o algoritmo só será vantajoso na multiplicação de matrizes muito grandes. Admitindo que o custo de cada multiplicação seja m, e de cada adição a, a fórmula do custo do algoritmo de Strassen é $7^k m + 6(7^k - 4^k)a$, para cal-

cular o produto de duas matrizes $2^k \times 2^k$, enquanto que o algoritmo clássico terá custo total $8^k m + (8^k - 4^k)a$[3]. O algoritmo começa a economizar tanto no número de somas como no número de multiplicações a partir de $k = 14$, isto é $n = 2^{14}$. O número total de operações no anel, (que é proporcional ao custo se $m = a$), começa a ser melhor para o algoritmo de Strassen a partir de $k = 10$, isto é, $n = 1.024$. Na prática, portanto, em muitos casos será mais conveniente continuar a usar o algoritmo clássico para calcular produtos de matrizes de tamanho relativamente pequeno, embora existam métodos de melhorar o desempenho do algoritmo de Strassen, além do indicado aqui. O artigo de P. Fischer mencionado no fim do capítulo apresenta alguns destes métodos e discute as condições em que o algoritmo de Strassen é superior ao algoritmo clássico.

Em geral, pelo mesmo método, utilizando recursivamente um algoritmo que, para um r dado, multiplica duas matrizes $r \times r$ usando m multiplicações, podemos obter um algoritmo que multiplica duas matrizes $n \times n$ para um n arbitrário usando $O(n^{\log_r m})$ multiplicações e somas. Assim, se para algum r existir um tal algoritmo com $\log_r m < \log_2 7$, então a partir dele poderíamos obter um algoritmo assintoticamente melhor do que o de Strassen. Recentemente V. Ya. Pan encontrou um tal algoritmo que multiplica matrizes $r \times r$ usando $m = (r^3 - 4r)/3 + 6r^2$ multiplicações. Em particular para $r = 70$ resulta $m = 143.640$, e como $\log_{70} 143.640 = 2,79512\ldots < \log_2 7$, seu algoritmo quando utilizado recursivamente é assintoticamente mais rápido do que o de Strassen. Talvez matrizes possam ser multiplicadas ainda mais rapidamente por métodos completamente diferentes, com possivelmente $n^2 \log n$ ou até mesmo n^2 multiplicações. Para que possamos ter uma idéia de quanta melhoria podemos ter alguma esperança de conseguir no futuro, é preciso que achemos limites inferiores para o número de operações necessárias. É este o assunto que discutiremos na Seção 4. Antes porém, veremos alguns outros problemas relacionados com o produto de matrizes.

---

(3) Estamos computando aqui apenas os custos devidos a operações no anel. Na realidade há um custo adicional de "controle" do processo, para tomar conta das chamadas recursivas e percorrer as matrizes da maneira correta. Este custo poderá ser desprezível ou não, dependendo do anel considerado, e dos métodos de acesso às matrizes.

3. Problemas correlatos

Nesta seção vamos mostrar que existem alguns problemas, cuja complexidade se relaciona com a do produto de matrizes, de tal modo que um algoritmo mais eficiente para o produto de matrizes também resulta em algoritmos melhores para estes problemas. A apresentação será algo esquemática, deixando detalhes e algumas demonstrações fáceis como exercícios. Assim, essa seção é de leitura algo mais difícil e pode ser omitida num primeiro estudo, sem prejuízo para a compreensão do restante do livro.

Seja A um anel com unidade. Um elemento $a \in A$ é *inversível* em A se existir um elemento indicado por a^{-1}, chamado de *inverso* de a em A tal que

$$a\,a^{-1} = a^{-1}a = 1.$$

É fácil ver que se um inverso de um elemento de A existir então ele é único. Dados $a, b \in A$, dizemos que eles *comutam* se $ab = ba$. Note que se b for inversível em A então a e b comutam sse a e b^{-1} também comutam. Neste caso definimos a operação de *divisão* de a por b, cujo resultado, o *quociente* entre a e b, é dado por

$$\frac{a}{b} = ab^{-1} = b^{-1}a.$$

Como a unidade comuta com todos os elementos de A, resulta que o inverso de qualquer elemento inversível de A pode ser obtido com uma operação de divisão pois $a^{-1} = 1/a$.

Estudaremos agora o número de multiplicações e/ou divisões em A envolvidas na determinação da inversa de uma matriz inversível no anel das matrizes $n \times n$ sobre A (vide Exercício 23(d)). Examinaremos apenas uma subclasse do conjunto das matrizes inversíveis. O método pode ser estendido a classes mais amplas conforme os Exercícios 10 e 11. Seja X uma matriz inversível $n \times n$ sobre A e escrevamos X como a matriz de blocos

$$X = \left[\begin{array}{c|c} X_{11} & X_{12} \\ \hline X_{21} & X_{22} \end{array}\right]$$

e suponhamos que a matriz X_{11} possui uma inversa X_{11}^{-1}, que n é

par, que todos os blocos têm dimensão $n/2 \times n/2$, e que a matriz

$$Y = X_{22} - X_{21}X_{11}^{-1}X_{12}$$

também é inversível, com inversa Y^{-1}. Então a inversa X^{-1} da matriz X pode ser escrita como

$$X^{-1} = \begin{bmatrix} X_{11}^{-1} + X_{11}^{-1}X_{12}Y^{-1}X_{21}X_{11}^{-1} & -X_{11}^{-1}X_{12}Y^{-1} \\ -Y^{-1}X_{21}X_{11}^{-1} & Y^{-1} \end{bmatrix}.$$

Podemos usar a identidade acima para calcular X^{-1}, usando duas inversões de matrizes $n/2 \times n/2$ e cinco produtos de matrizes $n/2 \times n/2$ (vide Exercício 9). Se n for uma potência de dois, isto sugere um método rápido de inversão de matrizes. Note, porém, que nem sempre é possível continuar o processo recursivamente: há matrizes inversíveis X tais que X_{11} ou Y não é inversível. Nos exercícios indicamos condições suficientes para que todas as submatrizes encontradas no processo sejam inversíveis, e, para matrizes de números reais, mostramos como reduzir o problema de inverter uma matriz inversível arbitrária, ao de inverter uma matriz que satisfaz estas condições. Assim, suporemos que a matriz X é inversível, n é potência de dois e que todas as submatrizes X_{11} e Y são inversíveis. Neste caso, usando as técnicas recursivas desenvolvidas para o método de Strasssen, temos a recorrência

$$\begin{cases} I(1) = 1, \\ I(n) = 2I\left(\dfrac{n}{2}\right) + 5\,M\left(\dfrac{n}{2}\right), \end{cases}$$

onde $I(n)$ denota o número de multiplicações e/ou divisões usadas pelo método para inverter matrizes $n \times n$, e $M(n)$ é o número de tais operações usadas no cálculo do produto de duas matrizes $n \times n$. Supondo que

$$\begin{cases} M(1) \geq 1, \\ M(n) \geq 4M\left(\dfrac{n}{2}\right), \end{cases} \tag{1}$$

Segue por indução que

$$I(n) \leq 3M(n).$$

Assim, é possível inverter matrizes deste tipo usando um número de multiplicações e/ou divisões da mesma ordem que o usado por qualquer algoritmo de multiplicação de matrizes, desde que a condição (1) seja satisfeita. Em particular, se usarmos o algoritmo de Strassen para a multiplicação de matrizes, a recorrência acima fica

$$\begin{cases} I(1) = 1, \\ I(n) = 2I\left(\frac{n}{2}\right) + 5\left(\frac{n}{2}\right)^{\log_2 7} \end{cases}$$

cuja solução é

$$I(n) = n^{\log_2 7}.$$

Reciprocamente, suponhamos agora que dispomos de um algoritmo que inverte matrizes $m \times m$ usando $I(m)$ operações de multiplicação e/ou divisão. Então, para calcular o produto de duas matrizes $n \times n$ X e Y considere a matriz $3n \times 3n$.

$$Z = \begin{bmatrix} I_n & X & 0_n \\ 0_n & I_n & Y \\ 0_n & 0_n & I_n \end{bmatrix}$$

onde I_n denota a matriz identidade $n \times n$ e 0_n a matriz $n \times n$ que tem todos os elementos nulos. É fácil ver que Z é inversível e Z^{-1} é dada por

$$Z^{-1} = \begin{bmatrix} I_n & -X & X \cdot Y \\ 0_n & I_n & -Y \\ 0_n & 0_n & I_n \end{bmatrix}$$

Como Z^{-1} pode ser obtido usando $I(3n)$ multiplicações e/ou divisões e ela contém a submatriz $X \cdot Y$, vemos que $I(3n)$ operações são suficientes também para o cálculo do produto de duas matrizes $n \times n$. Se denotarmos por $M(n)$ o número mínimo de multiplicações e/ou divisões necessário para multiplicar duas matrizes $n \times n$, isto mostra que

$$M(n) \leq I(3n).$$

Supondo que $I(3n) \in O(I(n))$ resulta que

$$M(n) \in O(I(n))$$

(para uma classe de funções que satisfazem a hipótese acima sobre $I(n)$ veja o Exercício 13 (c)).

Um outro problema relacionado com produto de matrizes é o de multiplicar matrizes booleanas. *Matrizes booleanas* são definidas sobre o conjunto $\{0, 1\}$ munido das operações $\vee$ (*disjunção*), $\wedge$ (*conjunção*) e $\neg$ (*negação*), definidas por:

$\vee$	0	1
0	0	1
1	1	1

$\wedge$	0	1
0	0	0
1	0	1

$\neg$	0	1
	1	0

Dadas duas matrizes booleanas $n \times n$ X e Y, seu *produto* é definido como a matriz booleana $Z = X \cdot Y$, cujos elementos z_{ij} são dados por

$$z_{ij} = \bigvee_{k=1}^{n} (x_{ik} \wedge y_{kj}) .$$

Matrizes booleanas podem ser utilizadas para representar relações sobre conjuntos finitos: dada uma relação R sobre o conjunto $\{1, 2, \ldots, n\}$, fazemos $r_{ij} = 1$ sse o elemento i está na relação R com o elemento j. É fácil ver que a relação representada pelo produto de matrizes booleanas é a composta das relações que as matrizes representam. Assim, o produto de matrizes booleanas é associativa, e a matriz booleana $n \times n$ I_n que representa a relação identidade sobre $\{1, 2, \ldots, n\}$ é a unidade em relação a este produto.

A *soma* das matrizes booleanas $n \times n$ X e Y, que indicaremos por $X \vee Y$, é a matriz booleana $n \times n$ W, cujos elementos w_{ij} são dados por

$$w_{ij} = x_{ij} \vee y_{ij}.$$

Definimos o *fecho reflexivo e transitivo* da matriz booleana $n \times n$ X como a matriz booleana $n \times n$ X^* dada pela soma

$$X^* = I_n \vee X \vee (X \cdot X) \vee ((X \cdot X) \cdot X) \vee \ldots .$$

A matriz booleana X^* representa o fecho reflexivo e transitivo R^* da relação R representada por X. (Veja os exercícios no fim deste capítulo e do Capítulo D. I.)

A seguir daremos algoritmos eficientes para o cálculo do fecho reflexivo e transitivo e para a multiplicação de matrizes booleanas. A medida de complexidade que adotaremos é o número de operações booleanas usado pelos algoritmos.

Em primeiro lugar, mostraremos que, sob certas hipóteses, os dois problemas têm complexidades assintóticas de mesma ordem. As reduções são muito semelhantes às vistas para a inversão de matrizes.

Suponhamos primeiro que temos um algoritmo que calcula o fecho reflexivo e transitivo de uma matriz booleana $m \times m$ usando $F(m)$ operações booleanas e que desejamos calcular o produto das matrizes booleanas $n \times n$ X e Y. Suponhamos ainda que $F(3n) \in O(F(n))$. Consideremos a matriz booleana $3n \times 3n$

$$Z = \begin{bmatrix} 0_n & X & 0_n \\ 0_n & 0_n & Y \\ 0_n & 0_n & 0_n \end{bmatrix}.$$

Então,

$$Z^2 = \begin{bmatrix} 0_n & 0_n & X \cdot Y \\ 0_n & 0_n & 0_n \\ 0_n & 0_n & 0_n \end{bmatrix}$$

e $Z^i = 0_{3n}$ para $i > 2$, onde 0_k é a matriz booleana $k \times k$ que tem todos os elementos nulos. Portanto, o fecho reflexivo e transitivo Z^* é dado por

$$Z^* = \begin{bmatrix} I_n & X & X \cdot Y \\ 0_n & I_n & Y \\ 0_n & 0_n & I_n \end{bmatrix},$$

e o produto $X \cdot Y$ aparece como uma submatriz de Z^*. Denotando por $B(n)$ o número mínimo de operações booleanas necessário para o cálculo do produto de matrizes booleanas $n \times n$, a construção acima mostra que

$$B(n) \leq F(3n),$$

e portanto, pela hipótese sobre F temos que $B(n) \in O(F(n))$.

Suponhamos agora que dispomos de um algoritmo de multiplicação de matrizes booleanas $n \times n$ usando $B(n)$ operações booleanas e que desejamos calcular o fecho reflexivo e transitivo de uma matriz

booleana X. Para simplificar o algoritmo, suporemos que n é uma potência de dois. Escrevamos X como a matriz de blocos $n/2 \times n/2$

$$X = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix}.$$

Então X^* é dado por

$$X^* = \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix},$$

onde os blocos Y_{11}, Y_{12}, Y_{21} e Y_{22} são dados por

$$\begin{aligned} Y_{11} &= (X_{11} \vee (X_{12} \cdot X_{22}^* \cdot X_{21}))^* \\ Y_{12} &= Y_{11} \cdot X_{12} \cdot X_{22}^* \\ Y_{21} &= X_{22}^* \cdot X_{21} \cdot Y_{11} \\ Y_{22} &= X_{22}^* \vee (X_{22}^* \cdot X_{21} \cdot Y_{11} \cdot X_{12} \cdot X_{22}^*) \end{aligned}$$

(vide Exercício 15). Assim, para calcular Y_{11}, Y_{12}, Y_{21} e Y_{22} por estas fórmulas, usamos dois fechos reflexivos e transitivos, seis produtos e duas somas, todas operações sobre matrizes booleanas $n/2 \times n/2$. Temos portanto, para o número $F(n)$ de operações booleanas usadas por este método, que

$$F(n) \leq 2F\left(\frac{n}{2}\right) + 6B\left(\frac{n}{2}\right) + 2\left(\frac{n}{2}\right)^2,$$

já que m^2 operações booleanas são suficientes para o cálculo da soma de duas matrizes booleanas $m \times m$. Supondo que a função $B(n)$ satisfaça

$$\begin{cases} B(1) \geq 1, \\ B(n) \geq 4B\left(\frac{n}{2}\right), \end{cases}$$

resulta, por indução, que

$$F(n) \leq 5B(n).$$

Agora veremos métodos para o cálculo do produto de matrizes booleanas (e, pelo visto acima, para o fecho reflexivo e transitivo).

É imediato verificar que n^3 operações $\wedge$ e $n^3 - n^2$ operações $\vee$ são suficientes para o cálculo do produto de duas matrizes booleanas, já que o algoritmo sugerido pela definição (análogo ao algoritmo clássico de multiplicação de matrizes) tem essa complexidade. Infelizmente as técnicas de Strassen, vistas na seção anterior, não se aplicam diretamente para obter um algoritmo eficiente, pois o conjunto $\{0, 1\}$ com as operações $\vee$ e $\wedge$ não é um anel, já que não existe o simétrico do elemento 1 para a operação $\vee$. (Veja os Exercícios 24 a 26 para o estudo de semianéis completos.) Apesar disto, é possível obter um algoritmo mais eficiente para a multiplicação de matrizes booleanas, como veremos a seguir.

Sejam X e Y duas matrizes booleanas $n \times n$. A matriz W, produto usual das matrizes X e Y consideradas como matrizes sobre o o anel dos inteiros, é uma matriz cujos elementos w_{ij} serão inteiros não-negativos. Seja Z a matriz booleana que é o produto (booleano) das matrizes booleanas X e Y. Então é fácil ver que $w_{ij} \neq 0$ sse $z_{ij} \neq 0$. A matriz W pode ser obtida em $O(n^{\log_2 7})$ operações aritméticas usando-se o algoritmo de Strassen. De posse de W, é fácil calcular Z.

O método acima descrito obtém o produto em $O(n^{\log_2 7})$ operações aritméticas. Estamos interessados no número de operações booleanas. Para transformar o algoritmo acima em um que use apenas operações booleanas ($\wedge$, $\vee$ e $\neg$) devemos implementar as operações aritméticas por meio de operações booleanas. É bem conhecido que a soma e o produto de números naturais pode ser calculado usando-se operações booleanas — é assim que os computadores modernos efetuam as quatro operações! Os algoritmos usuais têm complexidade $O(k)$ para a soma e $O(k^2)$ para o produto de dois números binários de k dígitos (vide Exercício 16). Para estimar o número de operações booleanas suficientes para o nosso algoritmo precisamos obter um limite superior para o número de dígitos suficiente para representar cada número usado no cálculo de W. Como cada elemento w_{ij} de W resulta da soma de n termos, cada um dos quais é no máximo 1, temos que $w_{ij} \leq n$, onde n é a dimensão da matriz W. Note que isto não significa que no cálculo dos w_{ij} pelo método de Strassen não possam surgir resultados intermediários cujo valor excede a n (vide Exercício 17). Para evitar a perda de eficiência que resultaria das operações com estes números grandes, podemos efetuar todas as somas e produtos módulo $n + 1$: isto corresponde a calcular o produto das matrizes inteiras X e Y sobre o anel $\mathbb{Z}_{n+1}$ dos inteiros módulo $n + 1$.

Como $\mathbb{Z}_{n+1}$ é um anel, o algoritmo de Strassem pode ser usado. Não é difícil provar que o resultado final será a mesma matriz W que seria obtida efetuando-se o produto no anel dos inteiros. Lembrando que o número $x \in \mathbb{N}$ pode sempre ser representado por $\lfloor log_2 x \rfloor + 1$ dígitos[4], vemos que se calcularmos W em $\mathbb{Z}_{n+1}$, nenhum resultado intermediário necessitará mais do que $\lfloor log_2 n \rfloor + 1$ dígitos.

Usando todas estas técnicas, obtemos um algoritmo que calcula o produto de duas matrizes booleanas com $O(n^{log_2 7} (log_2 n)^2)$ operações booleanas. De modo geral, se $M(n)$ for o número de operações aritméticas necessárias para o cálculo do produto de matrizes $n \times n$, e se $P(k)$ for um limite superior para o cálculo da soma e do produto de dois números de k dígitos, então $O(M(n)P(log_2 n))$ operações booleanas serão suficientes para o cálculo do produto de duas matrizes booleanas $n \times n$. Como vimos, este também é o número de operações suficiente para o cálculo do fecho reflexivo e transitivo de uma matriz booleana $n \times n$.

Existem outros problemas relacionados com multiplicação de matrizes: alguns são vistos nos exercícios. Mencionamos dois outros, que fogem do escopo deste livro:

(a) decomposição LUP: dada uma matriz $n \times n$ não singular X sobre um anel qualquer, é possível achar em $O(n^{2,795\ldots})$ operações aritméticas matrizes L, U e P, tais que $X = L \cdot U \cdot P$, P é uma matriz de permutação, L é uma matriz triangular inferior, U é uma matriz triangular superior.

(b) reconhecimento de linguagens livres de contexto: para cada gramática livre de contexto, é possível obter um algoritmo que, dada uma palavra x, decide se x está na linguagem gerada pela gramática em $O(n^{2,795\ldots})$ operações, onde n é o comprimento de x.

A seguir trataremos do problema de estabelecer limites inferiores para a complexidade do cálculo do produto de matrizes, e de alguns outros problemas algébricos.

4. Limites inferiores

Como já mencionamos no Capítulo I, é em geral muito difícil provar que uma determinada tarefa necessita de certos recursos com-

(4) $\lfloor x \rfloor$ é o maior inteiro menor ou igual a x.

putacionais, como um número mínimo de multiplicações, de comparações, tempo, espaço, etc. A dificuldade vem do fato de que para se provar um limite inferior de algum recurso computacional, devemos demonstrar que qualquer algoritmo que calcule a função desejada deverá usar pelo menos esta quantidade de recursos. É muitas vezes possível, e às vezes até fácil, mostrar que um particular algoritmo que conhecemos para a função utilize certos recursos. Mas para provar um limite inferior, este deve aplicar-se a métodos que calculem a função, mesmo a métodos que ainda não conhecemos, utilizando possivelmente propriedades ainda não descobertas. Precisamos enfim provar que *qualquer* algoritmo para a função terá de utilizar pelo menos tais e tais recursos. O exemplo da Seção 2 é um caso típico: antes de sua descoberta acreditava-se, erroneamente, que qualquer método para o cálculo do produto de duas matrizes $n \times n$ precisa utilizar pelo menos n^3 multiplicações. Consideremos agora um outro exemplo deste tipo. Suponha que queremos determinar o produto de dois números complexos. As operações permitidas são soma e multiplicação de números reais e queremos minimizar o número de multiplicações. Como $(x + iy)\ (u + iv) = (xu - yv) + i\,(xv + yu)$, aparentemente quatro multiplicações são necessárias. O seguinte algoritmo, no entanto, usa apenas três multiplicações.

procedimento *Prodcomplexo* (x, y, u, v):
  início
    $t_1, t_2, t_3 \leftarrow x + y,\ v - u,\ u + v$;
    $p_1 \leftarrow t_1 \cdot u$;
    $real \leftarrow p_1 - y \cdot t_3$;
    imaginária $\leftarrow x \cdot t_2 + p_1$;
    devolva (*real*, *imaginária*) $\{(x+iy)(u+iv) = (real + i \cdot imaginária)\}$
  fim

No caso de problemas algébricos, como produto de matrizes, cálculo de polinômios, e outros, existem algumas técnicas, que apesar das dificuldades apontadas, podem ser usadas para obter limites inferiores para o número de operações necessárias. Infelizmente, não será possível apresentarmos estas técnicas com detalhe e rigor suficientes no espaço limitado de que dispomos. Esperamos que as páginas seguintes dêem uma idéia do tipo de resultados que se pode obter e das técnicas utilizadas para prová-los, e que os leitores interessados consultem a bibliografia indicada. Vale a pena salientar que esta é uma área de pesquisa em franco desenvolvimento. Assim, é possível que

muitos dos resultados obtidos até agora sejam passíveis de melhoras substanciais.

Começaremos por um exemplo simples destas técnicas.

PROPOSIÇÃO 1. *Seja A um anel e seja*

$$f_n(a_0, a_1, \ldots, a_n) = \sum_{0 \le i \le n} a_i,$$

onde $a_i \in A$. *Qualquer programa, cujas únicas instruções sejam as operações do anel, e que calcule a função* f_n *requer n somas ou subtrações.*

Demonstração. Por indução em n. Se $n = 0$ então não há nada a provar. Suponhamos então que o cálculo de $f_n(a_0, a_1, \ldots, a_n)$ requer n operações $+$ ou $-$. Seja P um programa que calcula

$$f_{n+1}(a_0, a_1, \ldots, a_{n+1}) = \sum_{0 \le i \le n+1} a_i,$$

usando o número mínimo de somas e subtrações. O programa P deve ter pelo menos uma operação $+$ ou $-$ pois se todas as operações de P são multiplicações então somente valores da forma

$$c_1 \cdot a_{i_1}^{j_1} \cdot c_2 \cdot a_{i_2}^{j_2} \cdot \ldots \cdot c_k \cdot a_{i_k}^{j_k} \cdot c_{k+1}$$

podem ser calculadas. Em particular se $a_0 = a_1 = \ldots = a_{n-1} = 0$, então $f_{n+1}(a_0, a_1, \ldots, a_{n+1}) = a_n + a_{n+1}$, que para um anel arbitrário não pode ser expresso como um produto no anel. Consideremos então a primeira soma ou subtração de P. Todos os valores até agora calculados são produtos da forma já vista, pois esta é a primeira soma ou subtração utilizada. Esta soma ou subtração, então, precisa ser ou da forma $t_1 \pm t_2$ onde t_1 e t_2 são termos previamente calculados, e portanto da forma

$$c_1 \cdot a_{i_1}^{j_1} \cdot c_2 \cdot a_{i_2}^{j_2} \cdot \ldots \cdot c_k \cdot a_{i_k}^{j_k} \cdot c_{k+1}$$

ou uma variável simples, a_i, que é também da forma indicada. A soma ou subtração então equivale a uma instrução da forma

$$t \leftarrow c_1 \cdot a_{i_1}^{j_1} \cdot c_2 \cdot a_{i_2}^{j_2} \cdot \ldots \cdot c_k \cdot a_{i_k}^{j_k} \cdot c_{k+1} \pm d_1 \cdot a_{g_1}^{h_1} \cdot d_2 \cdot a_{g_2}^{h_2} \cdot \ldots \cdot d_e \cdot a_{g_e}^{h_e} d_{e+1}.$$

Pelo menos um dos termos t_1 ou t_2 precisa conter uma variável a_i, que suporemos sem perda de generalidade ser a_{n+1}. Seja P' o pro-

grama obtido de P onde todas as ocorrências de a_{n+1} são substituídas por 0. Tal programa calcula

$$f_n(a_0, a_1, \ldots, a_n).$$

A instrução correspondente à primeira soma ou subtração de P pode agora ser eliminada, pois o termo em que a_{n+1} aparece será igual a zero, e portanto esta instrução atribui a t ou o valor de uma variável, ou o valor de um termo anterior já calculado.[5] O programa assim obtido, P'', é equivalente a P', e calcula portanto $f_n(a_0, a_1, \ldots, a_n)$, e tem exatamente uma soma ou subtração a menos que P. Pela hipótese indutiva P'' precisa ter ao menos n somas ou subtrações. Conseqüentemente P precisa ter pelo menos n + 1 somas ou subtrações, o que prova a Proposição. ■

A demonstração acima ilustra o tipo de raciocínio utilizado neste ramo da teoria. Nos parágrafos seguintes elaboraremos os mecanismos necessários para provar limites inferiores do número de multiplicações necessárias para calcular o produto de duas matrizes.

Seja A um corpo,[6] e $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m$ variáveis que não sejam elementos de A. Considere o anel $A[x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m]$, o menor anel com unidade comutativo que contém A e as variáveis $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m$ e que estenda o corpo A. (Isto é, o anel de polinômios sobre A nas $n + m$ variáveis $x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m$.) Portanto, para elementos a e b de A, as operações $a + b$, $a - b$, $a \cdot b$, no anel $A[x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m]$ coincidem com as operações correspondentes de A. A idéia intuitiva de incluir as variáveis x_i e y_j é a de representar desta forma variáveis de entrada do algoritmo. Para cada execução do algoritmo, que operará sobre A, x_i e y_j assumirão valores em A, porém, como o algoritmo deve funcionar para quaisquer valores assumidos pelas variáveis de entrada, queremos que a priori não tenhamos nenhuma relação com qualquer elemento fixo de A.

Note que, como todo corpo é um anel, se um algoritmo calcular uma função sobre um anel arbitrário, em particular calcula tal função sobre todo corpo. Assim, podemos estudar limites inferiores no nú-

---

(5) Naturalmente, ao se eliminar esta instrução, é preciso também substituir t pela variável equivalente já conhecida, em todas as demais instruções em que t apareça.

(6) Um *corpo* é um anel com unidade comutativo em que todos os elementos não nulos são inversíveis.

mero de operações de algoritmos que operam sobre um corpo, que tais limites inferiores serão também válidos para algoritmos operando sobre anéis. A recíproca não é verdadeira: um algoritmo pode calcular uma função sobre um corpo usando um número de operações menor do que o usado por qualquer algoritmo que calcule a função sobre um anel arbitrário. Por exemplo, $xy + yx$ pode ser calculada com uma só multiplicação se x e y pertencerem a um corpo, mas requer duas multiplicações se eles forem elementos de um anel arbitrário.

Os algoritmos que estudaremos serão de um tipo especial, que denominaremos de algoritmos em linha reta. Um *algoritmo em linha reta* é um algoritmo, no sentido definido na Parte A, sem instruções de controle como **enquanto**, **repita**, **se então senão**, etc. ou de chamadas de procedimento. O algoritmo consiste de uma seqüência finita de instruções de atribuição de valor do tipo $u \leftarrow v \; op \; w$ onde v e w são:

(a) ou um dos x_i ou y_j;
(b) ou uma constante, isto é, um elemento de A;
(c) ou uma variável u cujo valor foi previamente calculado.

A operação *op* é uma das operações $+$, $-$, ou $\cdot$, do anel $A[x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m]$. Como queremos provar limites inferiores no número de operações aritméticas, as instruções de controle não são em geral necessárias, pois em princípio para cada n fixo, pode-se "desenrolar" um conjunto de instruções a ser repetida n vezes por meio de uma operação de controle como **enquanto** ou **repita**, pelo artifício de repetir no programa as operações envolvidas o número requerido de vezes. Desta maneira, o conceito de algoritmo em linha reta elimina os problemas irrelevantes ao aspecto que queremos estudar. Os limites inferiores que provaremos serão para algoritmos deste tipo, mas são igualmente aplicáveis a qualquer algoritmo mais geral, desde que este possa ser reduzido por um artifício, como o indicado acima, a um algoritmo em linha reta equivalente. Doravante quando usarmos a palavra "algoritmo" neste capítulo, subentenderemos ser um algoritmo em linha reta.

Um algoritmo, assim, *calcula uma função* f: $A^{n+m} \rightarrow A^p$ se, após a execução do algoritmo, um determinado p-subconjunto das variáveis u à esquerda das suas instruções contiver os p componentes do valor da função $f(x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m)$ para cada valor das variáveis de entrada.

Nosso objetivo é calcular limites inferiores para o número de multiplicações necessárias no cálculo de algumas funções. Para isto

contaremos apenas multiplicações que não possam ser eliminadas (por exemplo, a multiplicação $2 \cdot x$ pode ser eliminada, pela soma $x + x$), que chamaremos de essenciais. Como estamos provando limites inferiores, podemos contar apenas o número de multiplicações essenciais, pois este é menor do que ou igual ao número de multiplicações. Em outras palavras, provaremos limites inferiores sobre multiplicações essenciais, e por conseguinte tais limites inferiores aplicam-se a qualquer multiplicação, essencial ou não.

Não contaremos multiplicações entre elementos de A, nem multiplicações em que ambos os operandos não dependem de alguma das variáveis de entrada x_i e y_j. Assim, uma multiplicação é *essencial* se ela é uma operação $u \leftarrow v \cdot w$, onde v e w são variáveis essenciais. Uma variável é *essencial* se for

ou (a) um dos x_i ou y_j,

ou (b) uma variável cujo valor foi anteriormente calculado numa instrução, em que pelo menos um dos operandos é uma variável essencial.

A seguir, indicaremos como obter um limite inferior para o número de multiplicações essenciais necessárias para o cálculo de $B\mathbf{x}$, onde $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ é o vetor coluna dos valores x_i, e B é uma matriz $p \times n$, cujos elementos são funções lineares dos y_i. Os limites serão válidos para programas em linha reta que calculem a função $B\mathbf{x}$ para qualquer corpo A. Assim, não podemos usar propriedades de corpos particulares, nem comandos condicionais para testar propriedades dos argumentos.

A razão pela qual estudaremos o número de multiplicações necessárias para o cálculo de $B\mathbf{x}$ é que muitas funções podem ser representadas desta forma.

EXEMPLO 1. Sejam $U = [u_{ij}]$ e $V = [v_{ij}]$ matrizes $n \times n$ sobre A.

A matriz $Z = UV$ pode ser obtida efetuando-se o produto da matriz $n^2 \times n^2$ B com o vetor $\mathbf{x}$, onde, na forma de blocos,

$$B = \begin{bmatrix} U & 0_n & 0_n & \ldots & 0_n \\ 0_n & U & 0_n & \ldots & 0_n \\ 0_n & 0_n & 0_n & \ldots & U \end{bmatrix}$$

e

$$\mathbf{x}^T = (v_{11}, v_{21}, \ldots, v_{n1}, v_{12}, v_{22}, \ldots, v_{n2}, \ldots, v_{1n}, v_{2n}, \ldots, v_{nn}).$$

Obtém-se então

$$(z_{11}, z_{21}, \ldots, z_{n1}, z_{12}, z_{22}, \ldots, z_{n2}, \ldots, z_{n1}, z_{n2}, \ldots, z_{nn})^T.$$

EXEMPLO 2. As partes real e imaginária do produto de dois números complexos $u + iv$ e $t + iw$ podem ser calculadas efetuando-se o produto

$$\begin{bmatrix} u & -v \\ v & u \end{bmatrix} \quad \begin{bmatrix} t \\ w \end{bmatrix}.$$

Considere o espaço vetorial (sobre o corpo A) $A^k[y_1, y_2, \ldots, y_m]$ das k-tuplas de $A[y_1, y_2, , y_m]$. Um conjunto $\{\alpha_i\}$ de r vetores de $A^k[y_1, y_2, \ldots, y_m]$ é *linearmente independente módulo A* sse

$$\sum_{i=1}^{r} c_i \alpha_i \in A^k; \qquad c_i \in A$$

implica que $c_i = 0$ para todo i. Se um conjunto de vetores não é linearmente independente módulo A, dizemos que ele é *linearmente dependente módulo A*. Em outras palavras, uma coleção de vetores é linearmente independente módulo A se não existe uma combinação linear dos vetores com coeficientes em A, nem todos nulos, que resulte num vetor em cujas componentes as variáveis y_i não aparecem.

Seja B uma matriz $r \times s$ com elementos em $A[y_1, y_2, \ldots, y_m]$. O *posto de linhas* de B *módulo* A é a cardinalidade do maior conjunto de linhas de B (consideradas como vetores de $A^s[y_1, y_2, \ldots, y_m]$) que é linearmente independente módulo A.

A ferramenta essencial para a demonstração dos resultados desta seção é o teorema que provamos agora.

TEOREMA 1. *Seja B uma matriz com elementos em* $A[y_1, y_2, \ldots, y_m]$ *e seja* $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$. *Se o posto de linhas de B módulo A é r, o cálculo de* $B\mathbf{x}$ *requer pelo menos r multiplicações essenciais.*

Demonstração. Em primeiro lugar, note que podemos supor, sem perda de generalidade, que B tem r linhas. (Caso contrário, considere a matriz B' formada por r linhas independentes de B. Todo cálculo de $B\mathbf{x}$ deve calcular $B'\mathbf{x}$, e, portanto, usar pelo menos tantas multiplicações quanto $B'\mathbf{x}$.)

Suponha agora que s multiplicações são suficientes para o cálculo de $B\mathbf{x}$, e sejam $t_1, t_2, \ldots, t_s$ as expressões calculadas por essas multiplicações. Como as únicas outras operações permitidas são somas, subtrações ou multiplicações não essenciais, os elementos de $B\mathbf{x}$ podem ser expressos como funções lineares dos t_i e das incógnitas x_j e y_k, com os coeficientes c_ℓ das funções lineares sendo elementos de A. Isto é, se

$$\mathbf{t} = (t_1, t_2, \ldots, t_s)^T$$

podemos escrever o produto $B\mathbf{x}$ como

$$B\mathbf{x} = D\mathbf{t} + \mathbf{u}, \tag{2}$$

onde os elementos d_{ij} da matriz $r \times s$ D estão no corpo A e o vetor $\mathbf{u}$ tem por componentes funções lineares das incógnitas x_i e y_j, isto é, para todo ℓ, $1 \leq \ell \leq r$, existem c_i, $c_j \in A$, tais que

$$u_\ell = c_0 + \sum_{i=1}^{n} c_i x_i + \sum_{j=1}^{m} c_j y_j.$$

Suponhamos agora que $s < r$. Então, como sabemos da Álgebra Linear, as linhas $\mathbf{d}_i$ de D são linearmente dependentes (como vetores de A^s), e portanto existem coeficientes w_i, $w_i \in A$, nem todos nulos, tais que

$$\sum_{i=1}^{r} w_i \mathbf{d}_i = 0.$$

Seja $\mathbf{w}$ o vetor não nulo $\mathbf{w} = (w_1, w_2, \ldots, w_r)$. Multiplicando ambos os lados da igualdade (2) por $\mathbf{w}$, temos

$$\mathbf{w}B\mathbf{x} = \mathbf{w}D\mathbf{t} + \mathbf{w}\mathbf{u} = \mathbf{w}\mathbf{u}.$$

Podemos escrever esta igualdade como

$$(\mathbf{w}B)\mathbf{x} = \mathbf{w}\mathbf{u}.$$

Já que as componentes de $\mathbf{w}$ estão em A, o lado direito é uma função linear de incógnitas com coeficientes em A, e portanto o lado esquerdo não poderá ter produtos de incógnitas. Isto só é possível se o resultado de multiplicar o vetor linha $\mathbf{w}$ por B for um vetor cujas componentes estão em A (porque a linhas distintas de $\mathbf{x}$ correspondem variáveis x_i distintas). Mas $\mathbf{w}B \in A^n$ equivale a dizer que as linhas de B são linearmente dependentes módulo A, e, portanto, o posto de linhas de B módulo A não é r, contrariando a hipótese.

Como a contradição resultou do fato de supormos que $s < r$, devemos ter $s \geq r$, o que prova o teorema. ■

COROLÁRIO 1. *Pelo menos n^2 multiplicações são necessárias para calcular o produto de duas matrizes $n \times n$.*

Demonstração. O posto de linhas módulo A da matriz B do Exemplo 1 é n^2. ■

Existem várias maneiras de generalizar o teorema anterior: pode-se provar versões mais fortes do teorema que dão limites inferiores sobre algoritmos em linha reta em que divisões também são permitidas; pode-se demonstrar que o posto de colunas de B módulo A também é um limite inferior para o número de multiplicações essenciais necessárias para o cálculo de $B\mathbf{x}$, e pode-se obter uma formulação geral de limites inferiores sobre o número de multiplicações necessárias para o cálculo de formas bilineares baseada no posto de um tensor de ordem três associado à forma bilinear. Não podemos expor estas técnicas neste capítulo, apenas mencionamos alguns dos limites inferiores que podem ser provados por meio delas. Alguns destes tópicos são vistos nos exercícios no fim do capítulo.

Citamos os seguintes limites inferiores conhecidos para o número de multiplicações necessárias para o cálculo de algumas funções por algoritmos em linha reta:

cálculo do valor de um polinômio de grau n num ponto	n
produto de números complexos usando produtos de reais	3
produto de quaternions, usando produtos de reais	8
produto de matrizes quadradas $n \times n$	$2n^2 - n$.

Nos três primeiros casos, o limite superior é também o limite inferior: a regra de Horner usa n multiplicações para o cálculo do valor de um polinômio; vimos um algoritmo que calcula o produto de números complexos usando apenas 3 multiplicações; e o algoritmo de "multiplicação rápida" de quaternions pode ser encontrado em [66]. Produto de matrizes, no entanto, é um problema que continua em aberto: vimos que, até hoje, o melhor limite superior é da ordem de $n^{2,795\ldots}$ multiplicações, enquanto o melhor limite inferior é da ordem de n^2. A diferença entre os dois é muito grande, e é um problema fascinante (e aparentemente difícil) tentar diminuí-la.

EXERCÍCIOS

1. Verifique que o algoritmo *Prodmat2* calcula o produto $X \cdot Y$ corretamente, usando 7 multiplicações (não comutativas) e 18 somas.
2. Prove que o produto de matrizes pode ser efetuado por blocos.
3. (a) Prove que a solução da recorrência

$$\begin{cases} M(2^1) = 7 \\ M(2^k) = 7M(2^{k-1}) \text{ para } k > 1 \end{cases}$$

é $M(2^k) = 7^k$.

(b) Prove que a solução da recorrência

$$\begin{cases} A(2^1) = 18 \\ A(2^k) = 7 \cdot A(2^{k-1}) + 18(2^{k-1})^2 \text{ para } k > 1 \end{cases}$$

é $A(2^k) = 6 \cdot (7^k - 4^k)$.

4. Prove que a solução da recorrência

$$\begin{cases} X(1) = b, \\ X(n) = aX\left(\frac{n}{c}\right) + bn, \end{cases}$$

satisfaz

$$X(n) \in \begin{cases} O(n) & \text{se } a < c, \\ O(n \log n) & \text{se } a = c, \\ O(n^{\log_c a}) & \text{se } a > c. \end{cases}$$

5. Ache o menor n_0 tal que $6n_0^{\log_2 7} < n_0^3 - n_0^2$.
6. Mostre que o programa abaixo calcula o produto de duas matrizes 2×2 usando apenas 7 multiplicações e 15 somas (ou subtrações).

procedimento *Outroprodmat2*(X, Y):
início

$s_1 \leftarrow x_{21} + x_{22}$;
$s_2 \leftarrow s_1 - x_{11}$;
$s_3 \leftarrow x_{11} - x_{21}$;
$s_4 \leftarrow x_{12} - s_2$;
$s_5 \leftarrow y_{12} - y_{11}$;
$s_6 \leftarrow y_{22} - s_5$;
$s_7 \leftarrow y_{22} - y_{12}$;
$s_8 \leftarrow s_6 - y_{21}$;

$p_1 \leftarrow s_2 \cdot s_6$;
$p_2 \leftarrow x_{11} \cdot y_{11}$;
$p_3 \leftarrow x_{12} \cdot y_{21}$;
$p_4 \leftarrow s_3 \cdot s_7$;
$p_5 \leftarrow s_1 \cdot s_5$;
$p_6 \leftarrow s_4 \cdot y_{22}$;
$p_7 \leftarrow x_{22} \cdot s_8$;
$s_9 \leftarrow p_1 + p_2$;
$s_{10} \leftarrow s_9 + p_4$;
$z_{11} \leftarrow p_2 + p_3$;
$z_{12} \leftarrow s_9 + p_5 + p_6$;
$z_{21} \leftarrow s_{10} - p_7$;
$z_{22} \leftarrow s_{10} + p_5$;
devolva Z
fim

7. Escreva um programa em *LP* que calcula o produto de matrizes sobre $\mathbb{Z}_2$, o corpo dos inteiros módulo 2, sem usar a soma ou o produto de $\mathbb{Z}_2$.
8. Prove que uma multiplicação é necessária para o cálculo do produto de duas matrizes 1×1. (Sugestão: o programa deve funcionar para quaisquer anéis. Considere o anel dos polinômios com coeficientes em $\mathbb{Z}$ sobre duas variáveis não comutativas a e b, e mostre que o produto ab, resultado da multiplicação das matrizes 1×1 $[a]$ e $[b]$ não pode ser expresso como o resultado de somas e subtrações envolvendo a e b.)
9. Verifique a fórmula do texto para inversão de matrizes por blocos, e que ela pode ser programada usando apenas cinco multiplicações de matrizes.
10. Dê um exemplo de uma matriz inversível X, tal que
    (a) a submatriz X_{11} não seja inversível.
    (b) X_{11} seja inversível, mas $Y = X_{22} - X_{21} X_{11}^{-1} X_{21}$ não seja.
    Mostre que as matrizes X_{11} e Y são sempre inversíveis, se X for não-singular e
    (c) X é uma matriz triangular superior (triangular inferior).
    (d) A é um corpo ordenado e X é simétrica, positiva definida.

11. Mostre que se X é uma matriz não singular sobre um corpo ordenado, e se X^T é a transposta de X, então
    (a) $S = XX^T$ é simétrica positiva definida;
    (b) $X^{-1} = X^T \cdot S^{-1}$
    Usando os fatos acima, esboce um algoritmo para inverter matrizes sobre corpos ordenados que usa $O(M(n))$ multiplicações e divisões.

12. Prove que o método de Gauss para inversão de matrizes usa $O(n^3)$ operações de divisão e multiplicação.

13. (a) Prove que se $M(n) \geq 4M\left(\frac{n}{2}\right)$ então $I(n) \leq 3M(n)$ (Note que $M(1) \geq 1$ pelo Exercício 8.)
    (b) Prove que a solução de recorrência

$$\begin{cases} I(1) = 1, \\ I(n) = 2I\left(\frac{n}{2}\right) + 5 \cdot \left(\frac{n}{2}\right)^{\log_2 7}, \end{cases}$$

    satisfaz $I(n) < 7n^{\log_2 7}$. Obtenha a solução em seguida.
    (c) Mostre que a família de funções que sejam positivas, monotônicas não decrescentes, não limitadas e tais que $f(cn) \in \in O(f(n))$ para toda constante $c > 1$, contém as funções $a \cdot \log n$ e $a \cdot n^b$ para a e b reais e positivos. Ademais esta família é fechada sob soma, produto e composição de funções.

14. Mostre que
    (a) a operação $\wedge$ pode ser expressa por meio de composições de $\vee$ e $\neg$;
    (b) a operação $\vee$ pode ser expressa por meio de composições de $\wedge$ e $\neg$;
    (c) a operação $\neg$ não pode ser expressa por meio de composições de $\wedge$ e $\vee$.

15. Verifique a fórmula da Seção 3 para o fecho reflexivo e transitivo.

16. Dê algoritmos para a soma e para o produto de números naturais usando operações booleanas. Analise a complexidade dos algoritmos.

17. Considere a seguinte técnica para multiplicação de naturais escritos em notação binária:
    para multiplicar números de um dígito $x \cdot y = x \wedge y$;

para multiplicar números de n dígitos, onde $n = 2^k$, escreva

$$x = a \cdot 2^{n/2} + b$$
$$y = c \cdot 2^{n/2} + d$$

com a, b, c e d números de $n/2$ dígitos. Calcule $x \cdot y$ pelo algoritmo

$$u \leftarrow (a + b) \cdot (c + d);$$
$$v \leftarrow a \cdot c;$$
$$w \leftarrow b \cdot d;$$
$$z \leftarrow v \cdot 2^n + (u - v - w) \cdot 2^{n/2} + w$$

Lembrando que podemos multiplicar em notação binária por 2^n acrescentando n zeros à direita, vemos que o algoritmo acima usa três multiplicações de números de comprimento $n/2$.

(a) Escreva um algoritmo para multiplicação de inteiros usando a técnica acima. Mostre que sua complexidade satisfaz a recorrência

$$\begin{cases} P(1) = k, \\ P(n) = 3P(n/2) + kn, \end{cases}$$

onde k é uma constante.

(b) Mostre que a solução da recorrência acima é

$$P(n) = 3kn^{\log_2 3} - 2kn.$$

(c) O raciocínio acima contém um erro. Ache-o. Escreva as somas $(a + b)$ e $(c + d)$ como $\alpha 2^{n/2} + e$, e $\gamma 2^{n/2} + f$ com α e γ dígitos binários e obtenha um algoritmo correto. Mostre que sua complexidade é da ordem $n^{\log_2 3}$.

18. (a) Dê um exemplo de matrizes $n \times n$ sobre $\{0, 1\}$, cujo produto, pelo método de Strassen, produz resultados intermediários maiores do que n.

(b) Sejam X e Y matrizes de inteiros positivos, $Z = X \cdot Y$ e suponha que todos os elementos de X, Y e Z são limitados por n. Mostre que os resultados do produto $X \cdot Y$, calculados nos anéis $\mathbb{Z}$ e $\mathbb{Z}_{n+1}$ são idênticos. (Identificamos o inteiro $i \leq n$ com o elemento $1 + 1 + \ldots + 1$ (i vezes) de $\mathbb{Z}_{n+1}$.)

19. (a) Mostre que os vetores (y_1, y_2), $(y_2, 0)$ e $(0, y_2)$ de $A^2[y_1, y_2]$ são linearmente independentes módulo A.

(b) Defina o posto de colunas módulo A de uma matriz B com elementos em $A[y_1, y_2, \ldots, y_m]$. Mostre que o posto de colunas de B não é necessariamente igual ao posto de linhas de B.

20. Prove o seguinte teorema: Seja B uma matriz $p \times n$ cujos elementos são funções lineares das variáveis $y_1, y_2, \ldots, y_m$, e $\mathbf{x}$ um vetor coluna de n componentes. Se o posto de colunas de B módulo A for r, o cálculo de $B\mathbf{x}$ requer pelo menos r multiplicações essenciais.

21. Usando o teorema acima prove
    (a) o cálculo do valor de um polinômio de grau n num ponto requer n multiplicações (note que a regra de Horner usa exatamente este número de multiplicações e é, portanto ótima).
    (b) o cálculo do produto de uma matriz $n \times m$ por um vetor coluna de m componentes requer nm multiplicações.

    Sugestão: para a parte (a) considere o produto da matriz $1 \times (n+1)$ $[1\ x\ x^2\ \ldots\ x^n]$ pelo vetor $[a_0\ a_1\ \ldots\ a_n]^T$.
    Para a parte (b) considere o produto da matriz $n \times nm$

$$\begin{bmatrix} v_1 & v_2 & \ldots & v_m & 0 & 0 & \ldots & 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & v_1 & v_2 & \ldots & v_m & 0 & \ldots & 0 & 0 & \ldots & 0 \\ \multicolumn{14}{c}{\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots} \\ 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 & \ldots & v_1 & v_2 & \ldots & v_m \end{bmatrix}$$

    pelo vetor

$$[b_{11}\ b_{12}\ \ldots\ b_{1m}\ b_{21}\ b_{22}\ \ldots\ b_{2m}\ \ldots\ b_{n1}\ b_{n2}\ \ldots\ b_{nm}]^T.$$

22. Mostre como estender os algoritmos de inversão de matrizes e de fecho reflexivo e transitivo para o caso de matrizes $n \times n$, onde n não é uma potência de 2. Analise a complexidade dos algoritmos.

23. Considere o cálculo da função $p_n(x) = x^n$ sobre um corpo arbitrário.
    (a) mostre que se $n = 2^k$, então k multiplicações são suficientes para o cálculo de x^n.
    (b) mostre que se $\gamma(n)$ = número de algarismos 1 na representação binária de n, então $\lceil \log_2 n \rceil + \gamma(n) - 1$ multiplicações são suficientes para o cálculo de x^n.
    (c) mostre que se somente somas, subtrações e multiplicações são permitidas, então pelo menos $\lceil \log_2 n \rceil$ multiplicações são necessárias.
    (d) mostre que se divisões também são permitidas, então 6 multiplicações e/ou divisões são suficientes para calcular x^{31}, mas 7 multiplicações são necessárias em caso contrário.

24. Um *semianel completo* é um conjunto S, munido de operações de soma $+$ e produto $\cdot$ que satisfazem as seguintes propriedades
$(S, \cdot)$ é um monóide, isto é,
(i) $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ para todo $a, b, c \in S$;
(ii) Existe um elemento $1 \in S$ tal que

$$1 \cdot a = a \cdot 1 = a$$

para todo $a \in S$.
$(S, +)$ é um monóide comutativo, isto é,
(iii) $a + (b + c) = (a + b) + c$ para todo $a, b, c \in S$;
(iv) $a + b = b + a$ para todo $a, b \in S$;
(v) Existe um elemento $0 \in S$, tal que

$$a + 0 = 0 + a = a,$$

para todo $a \in S$.
A operação $\cdot$ distribui sobre $+$, isto é, para todo $a, b, c \in S$
(vi) $(a + b) \cdot c = a \cdot c + b \cdot c$;
(vii) $a \cdot (b + c) = a \cdot b + a \cdot c$.
O elemento 0 satisfaz
(viii) $a \cdot 0 = 0 \cdot a = 0$,
para todo $a \in S$.
A soma $\sum_{i \in I} a_i$ é definida para todo conjunto enumerável de índices I, com $a_i \in S$, e satisfaz
(ix) $\sum_{i \in \emptyset} a_i = 0$;
(x) $\sum_{i \in \{i\}} a_i = a_i$;
(xi) se $\{I_j \mid j \in J\}$ é uma partição de I, então

$$\sum_{i \in I} a_i = \sum_{j \in J} \left[\sum_{\ell \in I_j} a_\ell \right];$$

(xii) para todo $b \in S$

$$b \cdot \left[\sum_{i \in I} a_i \right] = \sum_{i \in I} b \cdot a_i.$$

Mostre que
(a) O conjunto $\{0, 1\}$ com as operações $\wedge$ e $\vee$ correspondendo a $\cdot$ e $+$, é um semianel completo.
(b) Seja $S = \mathbb{R}_+ \cup \{\infty\}$ onde $\mathbb{R}_+$ é o conjunto dos números reais não-negativos, com as operações de soma e *min* correspondendo a $\cdot$ e $+$ respectivamente. Então $(S, min, +)$ é um semianel completo.

(c) Seja S um semianel completo. Considere a família $M_n(S)$ das matrizes $n \times n$ com coeficientes em S. Mostre que $M_n(S)$ com as operações de soma e de produto de matrizes induzidas pelas operações $+$ e $\cdot$ de S, é um semianel completo.

(d) Mostre que

(d_1) as propriedades (iii) e (iv) da soma num semianel completo decorrem das propriedades (ix), (x) e (xi).

(d_2) a propriedade (viii) não é conseqüência das demais propriedades, mas seria conseqüência de (viii'): Para todo $a \in S$ existe um elemento $-a$, tal que $a + (-a) = (-a) + a = 0$. Mostre que a propriedade (viii') não vale nos semianéis (a) e (b) acima.

(d_3) Mostre que num semianel completo

$$\left[\sum_{i \in I} a_i\right] \cdot \left[\sum_{j \in J} b_j\right] = \sum_{i \in I,\, j \in J} a_i \cdot b_j .$$

25. Dado um semianel completo S, com as operações $+$ e $\cdot$, defina a^i para todo $a \in S$ e $i \in N$, por

$$\begin{cases} a^0 = 1, \\ a^{i+1} = a^i \cdot a. \end{cases}$$

Defina a^* como $a^* = \sum_{i \in N} a^i = 1 + a + a^2 + a^3 + \ldots$

No caso do semianel $M_n(S)$ do Exercício 24, a operação $*$ é chamada de fecho reflexivo e transitivo.

(a) Mostre que o produto de dois elementos de $M_n(S)$ pode ser obtido em $O(n^3)$ operações $+$ e $\cdot$ do semianel S.

(b) Mostre que num semianel completo S, o número $P(n)$ de operações $+$ e $\cdot$ de S necessárias para o cálculo do produto de duas matrizes de $M_n(S)$ é assintoticamente o mesmo que o número $F(n)$ de operações necessárias para o cálculo do fecho reflexivo e transitivo de um elemento de $M_n(S)$, supondo que $F(3n) \in \in O(F(n))$, $P(1) \geq 1$ e $P(n) \geq P(n/2)$. (Sugestão: a prova para o caso particular do semianel da parte (a) do Exercício 24 está no texto.)

26. (Cf. o Capítulo C. VI) Considere um grafo orientado cujas arestas são rotuladas com elementos de um semianel completo S. Seja M a matriz cujo elemento $m_{ij} = a$, $a \in S$ sse a soma dos rótulos das arestas com extremo inicial i e extremo final j é a. (E portanto, se não há aresta com extremo inicial i e extremo final j, então $m_{ij} = 0$.)

Dados dois vértices i e j e um passeio orientado P de arestas α_1, α_2, ..., α_k, rotulados por $a_1, a_2, \ldots, a_k$, o rótulo do passeio orientado P é o elemento $a_1 \cdot a_2 \cdot \ldots \cdot a_k$ de S.

(a) Verifique que o elemento m_{ij} de M é a soma dos rótulos de todos os passeios orientados de comprimento 1 de i a j.

(b) Verifique que o elemento (i, j) de M^2 é a soma dos rótulos de todos os passeios orientados de comprimento 2 de i a j.

(c) Verifique que o elemento (i, j) de M^* é a soma dos rótulos de todos os passeios orientados de i a j.

(d) Considere o caso particular do semianel $(\{0, 1\}, \vee, \wedge)$ do Exercício 24 (a). Mostre que o elemento (i, j) de M^* é $m^*_{ij} = 1$ sse existe um caminho orientado de i até j.

(e) Considere o semianel $(\mathbb{R}_+ \cup \{\infty\}, min, +)$ do Exercício 24 (b). Mostre que se o elemento (i, j) de M^* é d, então d é o comprimento de um caminho orientado de menor comprimento de i a j (onde o comprimento de cada aresta é igual ao seu rótulo).

NOTAS BIBLIOGRÁFICAS

O trabalho pioneiro na área é o artigo de Ostrowski [91] que coloca o problema de quantas multiplicações são necessárias para o cálculo de um polinômio num ponto, questão hoje totalmente esclarecida (cf. o Exercício 20 e o livro de Borodin e Munro [8]), embora um problema essencialmente equivalente ao do Exercício 22 (cadeias de adição) tenha sido anteriormente estudado em Matemática. O artigo de Strassen [116] expõe o método rápido de multiplicação de matrizes. A decomposição LUP mencionada no texto é do artigo [10] e o Exercício 10 é uma idéia não publicada de Schönhage. O Exercício 6 é devido a Winograd. O algoritmo de Pan a que referimos no texto está em [92]. A discussão do uso do algoritmo de Strassen é de [34]. O único livro-texto na área é [1]. Existe também uma monografia [8] que trata dos problemas de complexidade algébrica vistas no capítulo, expondo o que se sabia na área em 1975. Muitos dos resultados podem ser encontrados apenas como artigos de jornais.

Os resultados citados no texto sobre quaternions são de [66], o limite inferior de $2n^2 - n$ multiplicações para o produto de matrizes de [67] e o método rápido de multiplicação de inteiros é de [105].