obtained from $A$ by permuting its columns. Prove that the product of two permutation matrices is a permutation matrix. Prove that if $P$ is a permutation matrix, then $P$ is invertible, its inverse is $P^T$, and $P^T$ is a permutation matrix.

***31.1-3***
Prove that $(AB)^T = B^T A^T$ and that $A^T A$ is always a symmetric matrix.

***31.1-4***
Prove that if $B$ and $C$ are inverses of $A$, then $B = C$.

***31.1-5***
Let $A$ and $B$ be $n \times n$ matrices such that $AB = I$. Prove that if $A'$ is obtained from $A$ by adding row $j$ into row $i$, then the inverse $B'$ of $A'$ can be obtained by subtracting column $i$ from column $j$ of $B$.

***31.1-6***
Let $A$ be a nonsingular $n \times n$ matrix with complex entries. Show that every entry of $A^{-1}$ is real if and only if every entry of $A$ is real.

***31.1-7***
Show that if $A$ is a nonsingular symmetric matrix, then $A^{-1}$ is symmetric. Show that if $B$ is an arbitrary (compatible) matrix, then $BAB^T$ is symmetric.

***31.1-8***
Show that a matrix $A$ has full column rank if and only if $Ax = 0$ implies $x = 0$. (*Hint:* Express the linear dependence of one column on the others as a matrix-vector equation.)

***31.1-9***
Prove that for any two compatible matrices $A$ and $B$,

$$\operatorname{rank}(AB) \le \min(\operatorname{rank}(A), \operatorname{rank}(B)) ,$$

where equality holds if either $A$ or $B$ is a nonsingular square matrix. (*Hint:* Use the alternate definition of the rank of a matrix.)

***31.1-10***
Given numbers $x_0, x_1 \quad ., x_{n-1}$, prove that the determinant of the ***Vandermonde matrix***

$$V(x_0, x_1, \quad , x_{n-1}) = \begin{matrix} x_0 & x_0^2 & & x_0^{n-1} \\ x_1 & x_1^2 & & x_1^{n-1} \\ \vdots & & & \vdots \\ x_{n-} & x_{n-}^2 & & x_{n-1}^{n-1} \end{matrix} \Bigg)$$

is

$$\det(V(x_0, x_1, \quad ., x_{n-1})) = \prod_{0 \le j < k \le n-} (x_k - x_j)$$

(*Hint:* Multiply column $i$ by $-x_0$ and add it to column $i +$ for $i = n-1, n-2, \ldots, 1$, and then use induction.)

## 31.2 Strassen's algorithm for matrix multiplication

This section presents Strassen's remarkable recursive algorithm for multiplying $n \times n$ matrices that runs in $\Theta(n^{\lg 7}) = O(n^{2.81})$ time. For sufficiently large $n$, therefore, it outperforms the naive $\Theta(n^3)$ matrix-multiplication algorithm MATRIX-MULTIPLY from Section 26.1.

### An overview of the algorithm

Strassen's algorithm can be viewed as an application of a familiar design technique: divide and conquer. Suppose we wish to compute the product $C = AB$, where each of $A$, $B$, and $C$ are $n \times n$ matrices. Assuming that $n$ is an exact power of 2, we divide each of $A$, $B$, and $C$ into four $n/2 \times n/2$ matrices, rewriting the equation $C = AB$ as follows:

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix} . \tag{31.9}$$

(Exercise 31.2-2 deals with the situation in which $n$ is not an exact power of 2.) For convenience, the submatrices of $A$ are labeled alphabetically from left to right, whereas those of $B$ are labeled from top to bottom, in agreement with the way matrix multiplication is performed. Equation (31.9) corresponds to the four equations

$$\begin{aligned} r &= ae + bf , & (31.10) \\ s &= ag + bh , & (31.11) \\ t &= ce + df , & (31.12) \\ u &= cg + dh . & (31.13) \end{aligned}$$

Each of these four equations specifies two multiplications of $n/2 \times n/2$ matrices and the addition of their $n/2 \times n/2$ products. Using these equations to define a straightforward divide-and-conquer strategy, we derive the following recurrence for the time $T(n)$ to multiply two $n \times n$ matrices:

$$T(n) = 8T(n/2) + \Theta(n^2) . \tag{31.14}$$

Unfortunately, recurrence (31.14) has the solution $T(n) = \Theta(n^3)$, and thus this method is no faster than the ordinary one.

Strassen discovered a different recursive approach that requires only 7 recursive multiplications of $n/2 \times n/2$ matrices and $\Theta(n^2)$ scalar additions and subtractions, yielding the recurrence

$$T(n) = 7T(n/2) + \Theta(n^2) \tag{31.15}$$

$$\begin{aligned} &= \Theta(n^{\lg 7}) \\ &= O(n^{2.81}) . \end{aligned}$$

Strassen's method has four steps:

1. Divide the input matrices $A$ and $B$ into $n/2 \times n/2$ submatrices, as in equation (31.9).
2. Using $\Theta(n^2)$ scalar additions and subtractions, compute 14 $n/2 \times n/2$ matrices $A_1, B_1, A_2, B_2, \ldots, A_7, B_7$.
3. Recursively compute the seven matrix products $P_i = A_iB_i$ for $i = 1, 2, \ldots, 7$.
4. Compute the desired submatrices $r, s, t, u$ of the result matrix $C$ by adding and/or subtracting various combinations of the $P_i$ matrices, using only $\Theta(n^2)$ scalar additions and subtractions.

Such a procedure satisfies the recurrence (31.15). All that we have to do now is fill in the missing details.

## Determining the submatrix products

It is not clear exactly how Strassen discovered the submatrix products that are the key to making his algorithm work. Here, we reconstruct one plausible discovery method.

Let us guess that each matrix product $P_i$ can be written in the form

$$\begin{aligned} P_i &= A_iB_i \\ &= (\alpha_{i1}a + \alpha_{i2}b + \alpha_{i3}c + \alpha_{i4}d) \cdot (\beta_{i1}e + \beta_{i2}f + \beta_{i3}g + \beta_{i4}h) , \end{aligned} \quad (31.16)$$

where the coefficients $\alpha_{ij}$, $\beta_{ij}$ are all drawn from the set $\{-1, 0, 1\}$. That is, we guess that each product is computed by adding or subtracting some of the submatrices of $A$, adding or subtracting some of the submatrices of $B$, and then multiplying the two results together. While more general strategies are possible, this simple one turns out to work.

If we form all of our products in this manner, then we can use this method recursively without assuming commutativity of multiplication, since each product has all of the $A$ submatrices on the left and all of the $B$ submatrices on the right. This property is essential for the recursive application of this method, since matrix multiplication is not commutative.

For convenience, we shall use $4 \times 4$ matrices to represent linear combinations of products of submatrices, where each product combines one submatrix of $A$ with one submatrix of $B$ as in equation (31.16). For example, we can rewrite equation (31.10) as

$$\begin{aligned} r &= ae + bf \\ &\qquad \begin{matrix} a & b & c & d \end{matrix}) \begin{pmatrix} +1 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} \end{aligned}$$

$$= \begin{array}{c|cccc} & e & f & g & h \\ a & + & \cdot & \cdot & \cdot \\ b & & & & \\ c & & & & \\ d & \cdot & \cdot & \cdot & \cdot \end{array}$$

The last expression uses an abbreviated notation in which "+" represents $+1$, "·" represents 0, and "−" represents $-1$. (From here on, we omit the row and column labels.) Using this notation, we have the following equations for the other submatrices of the result matrix $C$:

$$\begin{aligned} s &= ag + bh \\ & \qquad\qquad + \\ &= \end{aligned}$$

$$\begin{aligned} t &= ce + df \\ &= \begin{pmatrix} \cdot & \cdot \\ \cdot & \cdot \\ + & \cdot \\ \cdot & + \end{pmatrix} \end{aligned}$$

$$\begin{aligned} u &= cg + dh \\ &= \Bigg( \end{aligned}$$

We begin our search for a faster matrix-multiplication algorithm by observing that the submatrix $s$ can be computed as $s = P_1 + P_2$, where $P_1$ and $P_2$ are computed using one matrix multiplication each:

$$\begin{aligned} P_1 &= A_1B_1 \\ &= a \cdot (g - h) \\ &= ag - ah \end{aligned}$$

$$\begin{aligned} P_2 &= A_2B_2 \\ &= (a + b) \cdot h \\ &= ah + bh \\ & \qquad\qquad + \\ &= \qquad\qquad + \end{aligned}$$

The matrix $t$ can be computed in a similar manner as $t = P_3 + P_4$, where

$$\begin{aligned} P_3 &= A_3B_3 \\ &= (c+d)\cdot e \\ &= ce + de \end{aligned}$$

and

$$\begin{aligned} P_4 &= A_4B_4 \\ &= d\cdot(f-e) \\ &= df - de \end{aligned}$$

\ - + · · /

Let us define an ***essential term*** to be one of the eight terms appearing on the right-hand side of one of the equations (31.10)–(31.13). We have now used 4 products to compute the two submatrices $s$ and $t$ whose essential terms are $ag$, $bh$, $ce$, and $df$. Note that $P_1$ computes the essential term $ag$, $P_2$ computes the essential term $bh$, $P_3$ computes the essential term $ce$, and $P_4$ computes the essential term $df$. Thus, it remains for us to compute the remaining two submatrices $r$ and $u$, whose essential terms are the diagonal terms $ae$, $bf$, $cg$, and $dh$, without using more than 3 additional products. We now try the innovation $P_5$ in order to compute two essential terms at once:

$$\begin{aligned} P_5 &= A_5B_5 \\ &= (a+d)\cdot(e+h) \\ &= ae + ah + de + dh \end{aligned}$$

( + · · + )

In addition to computing both of the essential terms $ae$ and $dh$, $P_5$ computes the inessential terms $ah$ and $de$, which need to be cancelled somehow. We can use $P_4$ and $P_2$ to cancel them, but two other inessential terms then appear:

$$P_5 + P_4 - P_2 = ae + dh + df - bh$$

( + · · · )

=

By adding an additional product

$$\begin{aligned} P_6 &= A_6B_6 \\ &= (b-d)\cdot(f+h) \\ &= bf + bh - df - dh \end{aligned}$$

+

\ /

however, we obtain

$$\begin{aligned} r &= P_5 + P_4 - P_2 + P_6 \\ &= ae + bf \end{aligned}$$

( + · · · )

+

We can obtain $u$ in a similar manner from $P_5$ by using $P_1$ and $P_3$ to move the inessential terms of $P_5$ in a different direction:

$$P_5 + P_1 - P_3 = ae + ag - ce + dh$$

( + · + · )

=

By subtracting an additional product

$$\begin{aligned} P_7 &= A_7B_7 \\ &= (a-c)\cdot(e+g) \\ &= ae + ag - ce - cg \end{aligned}$$

( + · + · )

=

we now obtain

$$\begin{aligned} u &= P_5 + P_1 - P_3 - P_7 \\ &= cg + dh \end{aligned}$$

The 7 submatrix products $P_1, P_2, \ldots, P_7$ can thus be used to compute the product $C = AB$, which completes the description of Strassen's method.

### Discussion

The large constant hidden in the running time of Strassen's algorithm makes it impractical unless the matrices are large ($n$ at least 45 or so) and dense (few zero entries). For small matrices, the straightforward algorithm is preferable, and for large, sparse matrices, there are special sparse-matrix algorithms that beat Strassen's in practice. Thus, Strassen's method is largely of theoretical interest.

By using advanced techniques beyond the scope of this text, one can in fact multiply $n \times n$ matrices in better than $\Theta(n^{\lg 7})$ time. The current best upper bound is approximately $O(n^{2.376})$. The best lower bound known is just the obvious $\Omega(n^2)$ bound (obvious because we have to fill in $n^2$ elements of the product matrix). Thus, we currently do not know how hard matrix multiplication really is.

Strassen's algorithm does not require that the matrix entries be real numbers. All that matters is that the number system form an algebraic ring. If the matrix entries do not form a ring, however, sometimes other techniques can be brought to bear to allow his method to apply. These issues are discussed more fully in the next section.

### Exercises

***31.2-1***
Use Strassen's algorithm to compute the matrix product

$$\begin{pmatrix} 1 & 3 \\ 5 & 7 \end{pmatrix} \begin{pmatrix} 8 & 4 \\ 6 & 2 \end{pmatrix}$$

Show your work.

***31.2-2***
How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which $n$ is not an exact power of 2? Show that the resulting algorithm runs in time $\Theta(n^{\lg 7})$.

***31.2-3***
What is the largest $k$ such that if you can multiply $3 \times 3$ matrices using $k$ multiplications (not assuming commutativity of multiplication), then you can multiply $n \times n$ matrices in time $o(n^{\lg 7})$? What would the running time of this algorithm be?

***31.2-4***
V. Pan has discovered a way of multiplying $68 \times 68$ matrices using 132,464 multiplications, a way of multiplying $70 \times 70$ matrices using 143,640 multiplications, and a way of multiplying $72 \times 72$ matrices using 155,424 multiplications. Which method yields the best asymptotic running time when

used in a divide-and-conquer matrix-multiplication algorithm? Compare it with the running time for Strassen's algorithm.

***31.2-5***
How quickly can you multiply a $kn \times n$ matrix by an $n \times kn$ matrix, using Strassen's algorithm as a subroutine? Answer the same question with the order of the input matrices reversed.

***31.2-6***
Show how to multiply the complex numbers $a + bi$ and $c + di$ using only three real multiplications. The algorithm should take $a$, $b$, $c$, and $d$ as input and produce the real component $ac - bd$ and the imaginary component $ad + bc$ separately.

## ★ 31.3 Algebraic number systems and boolean matrix multiplication

The properties of matrix addition and multiplication depend on the properties of the underlying number system. In this section, we define three different kinds of underlying number systems: quasirings, rings, and fields. We can define matrix multiplication over quasirings, and Strassen's matrix-multiplication algorithm works over rings. We then present a simple trick for reducing boolean matrix multiplication, which is defined over a quasiring that is not a ring, to multiplication over a ring. Finally, we discuss why the properties of a field cannot naturally be exploited to provide better algorithms for matrix multiplication.

### Quasirings

Let $(S, \oplus, \odot, \bar{0}, \bar{1})$ denote a number system, where $S$ is a set of elements, $\oplus$ and $\odot$ are binary operations on $S$ (the addition and multiplication operations, respectively), and $\bar{0}$ and $\bar{1}$ are distinct distinguished elements of $S$. This system is a ***quasiring*** if it satisfies the following properties:

1. $(S, \oplus, \bar{0})$ is a ***monoid***:
   - $S$ is ***closed*** under $\oplus$; that is, $a \oplus b \in S$ for all $a, b \in S$.
   - $\oplus$ is ***associative***; that is, $a \oplus (b \oplus c) = (a \oplus b) \oplus c$ for all $a, b, c \in S$.
   - $\bar{0}$ is an ***identity*** for $\oplus$; that is, $a \oplus \bar{0} = \bar{0} \oplus a = a$ for all $a \in S$.

   Likewise, $(S, \odot, \bar{1})$ is a monoid.
2. $\bar{0}$ is an ***annihilator***; that is, $a \odot \bar{0} = \bar{0} \odot a = \bar{0}$ for all $a \in S$.
3. The operator $\oplus$ is ***commutative***; that is, $a \oplus b = b \oplus a$ for all $a, b \in S$.
4. The operator $\odot$ ***distributes*** over $\oplus$; that is, $a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$ and $(b \oplus c) \odot a = (b \odot a) \oplus (c \odot a)$ for all $a, b, c \in S$.