4.1-4
Write pseudocode for a divide-and-conquer algorithm MATRIX-ADD-RECURSIVE that sums two $n \times n$ matrices A and B by partitioning each of them into four $n/2 \times n/2$ submatrices and then recursively summing corresponding pairs of submatrices. Assume that matrix partitioning uses $\Theta(1)$-time index calculations. Write a recurrence for the worst-case running time of MATRIX-ADD-RECURSIVE, and solve your recurrence. What happens if you use $\Theta(n^2)$-time copying to implement the partitioning instead of index calculations?

4.2 Strassen's algorithm for matrix multiplication

You might find it hard to imagine that any matrix multiplication algorithm could take less than $\Theta(n^3)$ time, since the natural definition of matrix multiplication requires n^3 scalar multiplications. Indeed, many mathematicians presumed that it was not possible to multiply matrices in $o(n^3)$ time until 1969, when V. Strassen [424] published a remarkable recursive algorithm for multiplying $n \times n$ matrices. Strassen's algorithm runs in $\Theta(n^{\lg 7})$ time. Since $\lg 7 = 2.8073549 \ldots$, Strassen's algorithm runs in $O(n^{2.81})$ time, which is asymptotically better than the $\Theta(n^3)$ MATRIX-MULTIPLY and MATRIX-MULTIPLY-RECURSIVE procedures.

The key to Strassen's method is to use the divide-and-conquer idea from the MATRIX-MULTIPLY-RECURSIVE procedure, but make the recursion tree less bushy. We'll actually increase the work for each divide and combine step by a constant factor, but the reduction in bushiness will pay off. We won't reduce the bushiness from the eight-way branching of recurrence (4.9) all the way down to the two-way branching of recurrence (2.3), but we'll improve it just a little, and that will make a big difference. Instead of performing eight recursive multiplications of $n/2 \times n/2$ matrices, Strassen's algorithm performs only seven. The cost of eliminating one matrix multiplication is several new additions and subtractions of $n/2 \times n/2$ matrices, but still only a constant number. Rather than saying "additions and subtractions"

everywhere, we'll adopt the common terminology of calling them both "additions" because subtraction is structurally the same computation as addition, except for a change of sign.

To get an inkling how the number of multiplications might be reduced, as well as why reducing the number of multiplications might be desirable for matrix calculations, suppose that you have two numbers x and y, and you want to calculate the quantity $x^2 - y^2$. The straightforward calculation requires two multiplications to square x and y, followed by one subtraction (which you can think of as a "negative addition"). But let's recall the old algebra trick $x^2 - y^2 = x^2 - xy + xy - y^2 = x(x - y) + y(x - y) = (x + y)(x - y)$. Using this formulation of the desired quantity, you could instead compute the sum $x + y$ and the difference $x - y$ and then multiply them, requiring only a single multiplication and two additions. At the cost of an extra addition, only one multiplication is needed to compute an expression that looks as if it requires two. If x and y are scalars, there's not much difference: both approaches require three scalar operations. If x and y are large matrices, however, the cost of multiplying outweighs the cost of adding, in which case the second method outperforms the first, although not asymptotically.

Strassen's strategy for reducing the number of matrix multiplications at the expense of more matrix additions is not at all obvious—perhaps the biggest understatement in this book! As with MATRIX-MULTIPLY-RECURSIVE, Strassen's algorithm uses the divide-and-conquer method to compute $C = C + A \cdot B$, where A, B, and C are all $n \times n$ matrices and n is an exact power of 2. Strassen's algorithm computes the four submatrices C_{11}, C_{12}, C_{21}, and C_{22} of C from equations (4.5)–(4.8) on page 82 in four steps. We'll analyze costs as we go along to develop a recurrence $T(n)$ for the overall running time. Let's see how it works:

1. If $n = 1$, the matrices each contain a single element. Perform a single scalar multiplication and a single scalar addition, as in line 3 of MATRIX-MULTIPLY-RECURSIVE, taking $\Theta(1)$ time, and return. Otherwise, partition the input matrices A and B and

output matrix C into $n/2 \times n/2$ submatrices, as in equation (4.2). This step takes $\Theta(1)$ time by index calculation, just as in MATRIX-MULTIPLY-RECURSIVE.

2. Create $n/2 \times n/2$ matrices $S_1, S_2, \ldots, S_{10}$, each of which is the sum or difference of two submatrices from step 1. Create and zero the entries of seven $n/2 \times n/2$ matrices $P_1, P_2, \ldots, P_7$ to hold seven $n/2 \times n/2$ matrix products. All 17 matrices can be created, and the P_i initialized, in $\Theta(n^2)$ time.

3. Using the submatrices from step 1 and the matrices $S_1, S_2, \ldots, S_{10}$ created in step 2, recursively compute each of the seven matrix products $P_1, P_2, \ldots, P_7$, taking $7T(n/2)$ time.

4. Update the four submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of the result matrix C by adding or subtracting various P_i matrices, which takes $\Theta(n^2)$ time.

We'll see the details of steps 2–4 in a moment, but we already have enough information to set up a recurrence for the running time of Strassen's method. As is common, the base case in step 1 takes $\Theta(1)$ time, which we'll omit when stating the recurrence. When $n > 1$, steps 1, 2, and 4 take a total of $\Theta(n^2)$ time, and step 3 requires seven multiplications of $n/2 \times n/2$ matrices. Hence, we obtain the following recurrence for the running time of Strassen's algorithm:

art

Compared with MATRIX-MULTIPLY-RECURSIVE, we have traded off one recursive submatrix multiplication for a constant number of submatrix additions. Once you understand recurrences and their solutions, you'll be able to see why this trade-off actually leads to a lower asymptotic running time. By the master method in Section 4.5, recurrence (4.10) has the solution $T(n) = \Theta(n^{\lg 7}) = O(n^{2.81})$, beating the $\Theta(n^3)$-time algorithms.

Now, let's delve into the details. Step 2 creates the following 10 matrices:

$$
\begin{aligned}
S_1 &= B_{12} - B_{22}, \\
S_2 &= A_{11} + A_{12}, \\
S_3 &= A_{21} + A_{22}, \\
S_4 &= B_{21} - B_{11}, \\
S_5 &= A_{11} + A_{22}, \\
S_6 &= B_{11} + B_{22}, \\
S_7 &= A_{12} - A_{22}, \\
S_8 &= B_{21} + B_{22}, \\
S_9 &= A_{11} - A_{21}, \\
S_{10} &= B_{11} + B_{12}.
\end{aligned}
$$

This step adds or subtracts $n/2 \times n/2$ matrices 10 times, taking $\Theta(n^2)$ time.

Step 3 recursively multiplies $n/2 \times n/2$ matrices 7 times to compute the following $n/2 \times n/2$ matrices, each of which is the sum or difference of products of A and B submatrices:

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 && (= A_{11} \cdot B_{12} - A_{11} \cdot B_{22}), \\
P_2 &= S_2 \cdot B_{22} && (= A_{11} \cdot B_{22} + A_{12} \cdot B_{22}), \\
P_3 &= S_3 \cdot B_{11} && (= A_{21} \cdot B_{11} + A_{22} \cdot B_{11}), \\
P_4 &= A_{22} \cdot S_4 && (= A_{22} \cdot B_{21} - A_{22} \cdot B_{11}), \\
P_5 &= S_5 \cdot S_6 && (= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}), \\
P_6 &= S_7 \cdot S_8 && (= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}), \\
P_7 &= S_9 \cdot S_{10} && (= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}).
\end{aligned}
$$

The only multiplications that the algorithm performs are those in the middle column of these equations. The right-hand column just shows

what these products equal in terms of the original submatrices created in step 1, but the terms are never explicitly calculated by the algorithm.

Step 4 adds to and subtracts from the four $n/2 \times n/2$ submatrices of the product C the various P_i matrices created in step 3. We start with

$$C_{11} = C_{11} + P_5 + P_4 - P_2 + P_6.$$

Expanding the calculation on the right-hand side, with the expansion of each P_i on its own line and vertically aligning terms that cancel out, we see that the update to C_{11} equals

which corresponds to equation (4.5). Similarly, setting

$$C_{12} = C_{12} + P_1 + P_2$$

means that the update to C_{12} equals

corresponding to equation (4.6). Setting

$$C_{21} = C_{21} + P_3 + P_4$$

means that the update to C_{21} equals

corresponding to equation (4.7). Finally, setting

$$C_{22} = C_{22} + P_5 + P_1 - P_3 - P_7$$

means that the update to C_{22} equals

which corresponds to equation (4.8). Altogether, since we add or subtract $n/2 \times n/2$ matrices 12 times in step 4, this step indeed takes $\Theta(n^2)$ time.

We can see that Strassen's remarkable algorithm, comprising steps 1–4, produces the correct matrix product using 7 submatrix

multiplications and 18 submatrix additions. We can also see that recurrence (4.10) characterizes its running time. Since Section 4.5 shows that this recurrence has the solution $T(n) = \Theta(n^{\lg 7}) = o(n^3)$, Strassen's method asymptotically beats the $\Theta(n^3)$ MATRIX-MULTIPLY and MATRIX-MULTIPLY-RECURSIVE procedures.

Exercises

Note: You may wish to read Section 4.5 before attempting some of these exercises.

4.2-1
Use Strassen's algorithm to compute the matrix product

Show your work.

4.2-2
Write pseudocode for Strassen's algorithm.

4.2-3
What is the largest k such that if you can multiply 3×3 matrices using k multiplications (not assuming commutativity of multiplication), then you can multiply $n \times n$ matrices in $o(n^{\lg 7})$ time? What is the running time of this algorithm?

4.2-4
V. Pan discovered a way of multiplying 68×68 matrices using 132,464 multiplications, a way of multiplying 70×70 matrices using 143,640 multiplications, and a way of multiplying 72×72 matrices using 155,424 multiplications. Which method yields the best asymptotic running time when used in a divide-and-conquer matrix-multiplication algorithm? How does it compare with Strassen's algorithm?

4.2-5
Show how to multiply the complex numbers $a + bi$ and $c + d\,i$ using only three multiplications of real numbers. The algorithm should take a,