Complexity and Cryptography

Thomas Zeugmann

Hokkaido University
Laboratory for Algorithmics

https://www-alg.ist.hokudai.ac.jp/~thomas/COCRB/

Lecture 2: Division and Matrix Multiplication

Division I

First, we have to clarify what we shall mean by division. Let numbers $a, d \in \mathbb{N}$ be given, where $d \neq 0$. There are two versions of division; i.e., computing the *quotient* a/d and computing *integers* q, r such that $a = qd + r$ and $0 \leqslant r < d$, respectively.

Division I

First, we have to clarify what we shall mean by division. Let numbers $a, d \in \mathbb{N}$ be given, where $d \neq 0$. There are two versions of division; i.e., computing the *quotient* a/d and computing *integers* q, r such that $a = qd + r$ and $0 \leqslant r < d$, respectively.

The second version (division with remainder) fits into the class of problems studied so far (n-bit numbers given as input are transformed into output numbers having $O(n^c)$ bits for a constant $c > 0$ independently of n).

Division II

The first version does not fit directly into the class of problems studied so far, since the quotient may have infinitely many bits. Thus, it is only meaningful to require the computation of a sufficiently precise approximation. Therefore, we first define what is meant by approximation.

Division II

The first version does not fit directly into the class of problems studied so far, since the quotient may have infinitely many bits. Thus, it is only meaningful to require the computation of a sufficiently precise approximation. Therefore, we first define what is meant by approximation.

Definition 1

Let x be any number. We say that $\tilde{x}$ is an *approximation of x with precision 2^{-c}* (precise for c bits) provided $|x - \tilde{x}| \leqslant 2^{-c}$.

Division III

Then, the division problem is defined as follows:

Division
Input: Numbers $a \in \mathbb{N}$, $d \in \mathbb{N}^+$ each having at most n bits.
Problem: Compute the quotient a/d with precision 2^{-n}.

Division III

Then, the division problem is defined as follows:

Division
Input: Numbers $a \in \mathbb{N}$, $d \in \mathbb{N}^+$ each having at most n bits.
Problem: Compute the quotient a/d with precision 2^{-n}.

Idea: We can split the problem into two subproblems; i.e., computing the inverse d^{-1} of d with precision 2^{-2n} and then multiplying this approximation of d^{-1} and a.
In the following, we use $\tilde{d}^{-1}$ to denote the approximation of d^{-1} with precision 2^{-2n}.

Division IV

Lemma 1 (Precision)

Let $\mathfrak{a}$, $\mathfrak{d}$ *be* $\mathfrak{n}$*-bit numbers. Suppose we have computed the inverse* $\mathfrak{d}^{-1}$ *of* $\mathfrak{d}$ *with precision* $2^{-2\mathfrak{n}}$. *Then* $\mathfrak{a}\tilde{\mathfrak{d}}^{-1}$ *is an approximation of* $\mathfrak{a}/\mathfrak{d}$ *with precision* $2^{-\mathfrak{n}}$.

Division IV

Lemma 1 (Precision)

Let a, d *be* n*-bit numbers. Suppose we have computed the inverse* d^{-1} *of* d *with precision* 2^{-2n}*. Then* $a\tilde{d}^{-1}$ *is an approximation of* a/d *with precision* 2^{-n}*.*

Proof. Let $\tilde{d}^{-1}$ be the approximation of d^{-1} with precision 2^{-2n}. By assumption, $a < 2^n$, and thus

$$\begin{aligned} \left| \frac{a}{d} - a\tilde{d}^{-1} \right| &= |a||d^{-1} - \tilde{d}^{-1}| \\ &\leqslant 2^n \cdot 2^{-2n} = 2^{-n} . \end{aligned}$$

∎

Division V

First, we deal with the computation of d^{-1}. One possible approach would be to use the fact

$$\frac{1}{1-x} = \sum_{i=0}^{\infty} x^i \quad \text{provided } |x| < 1 \,.$$

We can assume, without loss of generality, that $1/2 \leqslant d < 1$.

Division V

First, we deal with the computation of d^{-1}. One possible approach would be to use the fact

$$\frac{1}{1-x} = \sum_{i=0}^{\infty} x^i \quad \text{provided } |x| < 1 \,.$$

We can assume, without loss of generality, that $1/2 \leqslant d < 1$.

Proof. Let $d = \sum_{i=0}^{n-1} d_i 2^i$ and let d_k be the highest non-zero bit, i.e., $d_k \neq 0$, $k \leqslant n-1$ and $d_{k+1} = \cdots = d_{n-1} = 0$ provided $k < n-1$. Then, $\hat{d} =_{df} d2^{-(k+1)}$ satisfies $1/2 \leqslant \hat{d} < 1$. Thus, $\hat{d}$ can be computed from d by applying a simple shift operation. However, for determining k one needs $O(n)$ bit operations in the worst case.

Division VI

Now, since the inverse of $2^{-(k+1)}$ is 2^{k+1}, knowing the inverse of $\hat{d}$ with precision 2^{-2n} is all we need. So, we can set $x = 1 - d$.

It remains to ask how many summands we actually have to compute to achieve the desired approximation. The answer is provided by the following lemma:

Division VII

Lemma 2

Let d^{-1} *be the exact inverse of* d, *where* $1/2 \leqslant d < 1$. *Furthermore, let* $b_{2n} = \sum_{i=0}^{2n} (1-d)^i$. *Then, we have* $|d^{-1} - b_{2n}| \leqslant 2^{-2n}$.

Division VII

Lemma 2

Let d^{-1} *be the exact inverse of* d, *where* $1/2 \leqslant d < 1$. *Furthermore, let* $b_{2n} = \sum_{i=0}^{2n} (1-d)^i$. *Then, we have* $|d^{-1} - b_{2n}| \leqslant 2^{-2n}$.

Proof. First, since $1/2 \leqslant d < 1$ we can conclude that $0 < 1 - d \leqslant 1/2$. Thus, the geometrical series $\sum_{i=0}^{\infty} (1-d)^i$ is absolutely convergent and from calculus we know that

$$\sum_{i=0}^{\infty} (1-d)^i = \frac{1}{1-(1-d)} = \frac{1}{d}\,.$$

Therefore, $\sum_{i=0}^{\infty} (1-d)^i$ is the *exact* inverse of d.

Division VIII

Hence, we get

$$\begin{aligned}
|d^{-1} - b_{2n}| &= \left| \sum_{i=0}^{\infty} (1-d)^i - \sum_{i=0}^{2n} (1-d)^i \right| \\
&= \left| \sum_{i=2n+1}^{\infty} (1-d)^i \right| \\
&\leqslant \sum_{i=2n+1}^{\infty} |(1-d)^i| \leqslant \sum_{i=2n+1}^{\infty} \left(\frac{1}{2}\right)^i \\
&= \frac{1 - 1 + \left(\frac{1}{2}\right)^{2n+1}}{1 - \frac{1}{2}} = 2^{-2n}.
\end{aligned}$$

∎

Division via Newton I

However, the resulting algorithm is quite slow compared to multiplication.

Question

Can we do any better?

Division via Newton I

However, the resulting algorithm is quite slow compared to multiplication.

Question

Can we do any better?

The affirmative answer is obtained as follows: We can avoid to compute b_{2n+1} by evaluating the sum given above if we use the well-known Newton procedure for computing zeros of functions.

Division via Newton II

Recall that for a given differentiable function f and x_* with $f(x*) = 0$, one can compute x_* with any desired precision by using

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}\,, \tag{1}$$

provided x_0 is appropriately chosen. That is, then we know from calculus that

$$\lim_{k\to\infty} x_k = x_* \,.$$

Division via Newton II

Recall that for a given differentiable function f and x_* with $f(x*) = 0$, one can compute x_* with any desired precision by using

$$x_k = x_{k-1} - \frac{f(x_{k-1})}{f'(x_{k-1})}\,, \tag{1}$$

provided x_0 is appropriately chosen. That is, then we know from calculus that

$$\lim_{k\to\infty} x_k = x_* \,.$$

What is the appropriate f in our case?

Division via Newton III

In order to apply this method to our setting of computing the inverse, we *cannot* use $f(x) = xd - 1$. In this case, we would get $f'(x) = d$, and thus for computing the iteration in (1), we already should know d^{-1}.

Division via Newton III

In order to apply this method to our setting of computing the inverse, we *cannot* use $f(x) = xd - 1$. In this case, we would get $f'(x) = d$, and thus for computing the iteration in (1), we already should know d^{-1}.

So, we set $f(x) = d - 1/x$, and obtain thus from (1) that the sequence $(b_k)_{k\in\mathbb{N}}$ defined by

$$b_k = (2 - b_{k-1}d)b_{k-1} \tag{2}$$

converges to d^{-1} provided b_0 is appropriately chosen.

Division via Newton IV

Also, we should know that Newton's procedure converges quadratically. Thus, $O(\log n)$ iterations will suffice. For the sake of completeness, and for justifying our choice of b_0 we include the following theorem here:

Theorem 1 (Number of Newton Iterations)

Let d *be such that* $1/2 \leqslant d < 1$, *let* $b_0 = 1$ *and let* $b_k = (2 - b_{k-1}d)b_{k-1}$ *for all* $k \geqslant 1$. *Then we have*

$$b_k = \sum_{i=0}^{2^k-1} (1-d)^i .$$

Proof I

Proof. We prove the theorem by induction. For the induction base $k = 1$ we directly obtain

$$b_1 = (2 - b_0 d)b_0 = 1 + (1 - d) = \sum_{i=0}^{2^1-1} (1 - d)^i .$$

Proof I

Proof. We prove the theorem by induction. For the induction base $k = 1$ we directly obtain

$$b_1 = (2 - b_0 d)b_0 = 1 + (1 - d) = \sum_{i=0}^{2^1-1} (1 - d)^i .$$

The induction step from k to $k + 1$ is derived as follows: We have the induction hypothesis (abbr. IH)

$$b_k = \sum_{i=0}^{2^k-1} (1 - d)^i .$$

Proof II

$$
\begin{aligned}
b_{k+1} &= (2 - b_k d) b_k = 2b_k - b_k d b_k \\
&= 2b_k - \sum_{i=0}^{2^k-1} (1-d)^i \cdot d \cdot \sum_{i=0}^{2^k-1} (1-d)^i \quad (\text{ by the IH}) \\
&= 2b_k + \sum_{i=0}^{2^k-1} (1-d)^i (-1 + (1-d)) \sum_{i=0}^{2^k-1} (1-d)^i \\
&= 2b_k - \sum_{\mathrm{i}=0}^{2^k-1} (1-d)^i \sum_{i=0}^{2^k-1} (1-d)^i + \sum_{\mathrm{i}=0}^{2^k-1} (1-d)^{i+1} \sum_{i=0}^{2^k-1} (1-d)^i
\end{aligned}
$$

Proof III

$$
\begin{aligned}
&= 2b_k - b_k - \sum_{i=1}^{2^k-1} (1-d)^i \sum_{i=0}^{2^k-1} (1-d)^i + \sum_{i=1}^{2^k-1} (1-d)^i \sum_{i=0}^{2^k-1} (1-d)^i \\
&\quad + (1-d)^{2^k} \sum_{i=0}^{2^k-1} (1-d)^i \\
&= b_k + \sum_{i=0}^{2^k-1} (1-d)^{2^k+i} \\
&= \sum_{i=0}^{2^k-1} (1-d)^i + \sum_{i=2^k}^{2^{k+1}-1} (1-d)^i \quad \text{by the IH} \\
&= \sum_{i=0}^{2^{k+1}-1} (1-d)^i . \qquad \blacksquare
\end{aligned}
$$

Division XIII

The latter theorem essentially shows that the number of correct bits is doubled in each iteration.
Summarizing the results obtained so far, now we can prove the following theorem: In the following $M(n)$ denotes an upper bound on the time needed to do a multiplication of two n-bit numbers.

Division XIII

The latter theorem essentially shows that the number of correct bits is doubled in each iteration.
Summarizing the results obtained so far, now we can prove the following theorem: In the following $M(n)$ denotes an upper bound on the time needed to do a multiplication of two n-bit numbers.

Theorem 2

There is an algorithm which, on input any two numbers a and d having at most n bits, computes the quotient a/d with precision 2^{-n} using time $O(M(n) \log n)$.

Division XIV

Proof. First, using Theorem NNI we can compute the inverse d^{-1} of d with precision 2^{-2n} by using $\lceil \log 2n \rceil + 1$ many iterations. In each iteration we have to perform two multiplications and one addition. The two multiplications require time $O(M(n))$ and the addition needs time $O(n)$. Of course, we have to truncate the result of each iteration to the $2n$ leading bits.

Furthermore, by Lemma (Precision), it then suffices to multiply a and the approximate inverse. This requires another multiplication of two numbers having at most $2n$ bits. Thus, the overall time complexity is $O(M(n) \log n)$. ∎

Discussion I

Does this mean that division is more complex than multiplication?

Discussion I

Does this mean that division is more complex than multiplication?

We may be tempted to answer this question affirmatively, but some care has to be taken here. Of course, we can try to prove a lower bound on the number of bit operations needed to perform division. Provided we could show this lower bound to be $\Omega(M(n)\log n)$, we are done in the sense that we then know no improvement is possible. But this is much easier said than done. Proving non-trivial lower bounds is a very hard task (and we still have almost no experience in doing so). Second, we could try to find another method for division. But again, this is easier said than done. Nevertheless, we could try it.

Discussion II

Finally, it is well possible that we have already collected all good ideas needed, but failed to put them together in the right way. Maybe, we have been a bit too generous. The point here is that we always perform multiplications and additions of $2n$-bit numbers.

Discussion II

Finally, it is well possible that we have already collected all good ideas needed, but failed to put them together in the right way. Maybe, we have been a bit too generous. The point here is that we always perform multiplications and additions of $2n$-bit numbers.

Taking into account that the Newton procedure is *self-correcting*, Cook (1966) observed that we can ignore the bits that are anyhow not correct. That is, instead of performing multiplications and additions of $2n$-bit numbers we only use the highest 2^k bits in iteration k when executing (2).

Discussion III

We illustrate the idea below. In our example we set $d = 0.66$ and assume that we need the first 8 digits to be correct. Thus, we have to compute 1.5151515 (we perform the calculation here in decimal notion). The left column shows the results when always 8 digits are used, while the right column displays the result when using 2, 4, and 8 digits.

	8 digits	2^k digits
b_1	1.34	1.4
b_2	1.494904	1.506
b_3	1.5148809	1.5150962
b_4	1.5151514	1.5151515
b_5	1.5151515	1.5151515

This looks pretty good.

Division versus Multiplication I

Now we can establish the complexity of division relative to multiplication.

We make the following assumptions about the function $M(n)$: First, we assume $M(n) \geqslant n$ for all $n \in \mathbb{N}^+$ and that $M(m) \leqslant M(n)$ for all $m, n \in \mathbb{N}^+$ provided $m \leqslant n$. Second, we assume that $M(n) = ng(n)$, where $g\colon \mathbb{N} \to \mathbb{N}$ is a function satisfying $g(n) > 0$ and $g(m) \leqslant g(n)$ for all $m, n \in \mathbb{N}^+$ whenever $m \leqslant n$. Note that the given upper bounds $M(n)$ satisfy these assumptions.

Consequently, assuming $m \leqslant n$, we obtain

$$\begin{aligned} mg(m) + ng(n) &\leqslant mg(n) + ng(n) = (m+n)g(n) \\ &\leqslant (m+n)g(m+n)\,. \end{aligned}$$

This in turn implies that

$$M(m) + M(n) \leqslant M(m+n) \quad \text{for all } m, n \in \mathbb{N}^+\,. \tag{3}$$

Division versus Multiplication II

Let $I(n)$ denote the time needed to compute the inverse of any n-bit number with precision 2^{-2n}. Using the ideas just explained, we obtain the following theorem:

Division versus Multiplication II

Let $I(n)$ denote the time needed to compute the inverse of any n-bit number with precision 2^{-2n}. Using the ideas just explained, we obtain the following theorem:

Theorem 3

There exists a constant $c > 0$ *such that* $I(n) \leqslant c \cdot M(n)$ *for all* $n \in \mathbb{N}^+$.

Division versus Multiplication II

Let $I(n)$ denote the time needed to compute the inverse of any n-bit number with precision 2^{-2n}. Using the ideas just explained, we obtain the following theorem:

Theorem 3

There exists a constant $c > 0$ such that $I(n) \leqslant c \cdot M(n)$ for all $n \in \mathbb{N}^+$.

Proof. We execute the iteration given in (2), but use only the highest 2^k bits of d and b_{k-1} for computing b_k, where again $b_0 = 1$, and $k = 1, \ldots, \lceil \log 2n \rceil + 1$. The result is then truncated to the highest 2^{k+1} bits.

Division versus Multiplication III

Each iteration needs two multiplications and one addition. So, we can uper bound the complexity of iteration k by $\tilde{c} \cdot M(2^k)$ for a suitable constant $\tilde{c} > 0$. Using (3) we thus obtain

$$I(n) \leqslant \tilde{c} \cdot (M(2) + \cdots + M(2^{\log 2n+1})) \leqslant \tilde{c} \cdot M(\hat{c}n) .$$

Since $M(n) \leqslant n^2$, we have $\tilde{c} \cdot M(\hat{c}n) \leqslant \tilde{c} \cdot \hat{c}^2 M(n)$, and thus, it suffices to set $c = \tilde{c} \cdot \hat{c}^2$.
Now one has to show that the sequence $(b_k)_{k\in\mathbb{N}}$ computed in the way described above converges to the inverse of d and that we have to compute only the first $\lceil \log 2n \rceil + 1$ members of it.
This is left as an exercise. ∎

Matrix Multiplication I

Next, we turn our attention to another fundamental problem, i.e., matrix multiplication. Matrix multiplication is needed in numerous applications involving linear algebra. Therefore, studying the complexity of matrix multiplication is not only of theoretical interest but also of fundamental practical importance.

Matrix Multiplication I

Next, we turn our attention to another fundamental problem, i.e., matrix multiplication. Matrix multiplication is needed in numerous applications involving linear algebra. Therefore, studying the complexity of matrix multiplication is not only of theoretical interest but also of fundamental practical importance.

In order to achieve as much generality as possible, we shall consider matrix multiplications for any matrices defined over a ring.

Matrix Multiplication II

Definition 2

Let S be any non-empty set containing at least two distinguished elements 0 and 1, and let $+$ and $\cdot$ be binary operations over S (that is $+ : S \times S \to S$ and $\cdot : S \times S \to S$). Then $R = (S, +, \cdot, 0, 1)$ is a *ring* with identity element provided that for all $a, b, c \in S$ the following properties hold:

(1) $(a + b) + c = a + (b + c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$;

(2) $(a + b) = (b + a)$ ($+$ is commutative);

(3) $(a + b) \cdot c = a \cdot c + b \cdot c$ and $a \cdot (b + c) = a \cdot b + a \cdot c$;

(4) $a + 0 = 0 + a = a$ (0 is the neutral element with respect to $+$);

(5) $a \cdot 1 = 1 \cdot a = a$ (1 is the identity element with respect to $\cdot$);

(6) For each $a \in S$ there exist an element $-a \in S$ such that $a + (-a) = (-a) + a = 0$ ($-a$ is the additive inverse of a).

Matrix Multiplication III

Since we only consider rings with identity, we refer to a ring with identity as to a ring for short.
Furthermore, if R is a ring and the operation $\cdot$ is commutative, then we say that the ring R is *commutative*. Finally, if R is a commutative ring and if for every $a \in S \setminus \{0\}$ there exist an element $a^{-1} \in S$ such that $a \cdot a^{-1} = a^{-1} \cdot a = 1$, then R is said to be a *field*.

So, let $R = (S, +, \cdot, 0, 1)$ be a commutative ring with 1. Furthermore, let $n \in \mathbb{N}^+$. We consider the set M_n of all $n \times n$ matrices over R.

Matrix Multiplication IV

Moreover, let

$$I_n = \begin{pmatrix} 1 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & 1 & \cdot & \cdot & \cdot & 0 \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ 0 & 0 & \cdot & \cdot & 0 & 1 \end{pmatrix}$$

Matrix Multiplication V

and

$$0_n = \begin{pmatrix} 0 & \cdot & \cdot & \cdot & 0 \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ 0 & \cdot & \cdot & \cdot & 0 \end{pmatrix}$$

Matrix Multiplication VI

Addition and multiplication of matrices from M_n is defined as usual by using the addition and multiplication from the underlying ring R. We denote the resulting operations by $+_n$ and $\times_n$, respectively. That is, for $A = (a_{ij})$ and $B = (b_{ij})$, $i, j = 1, \ldots, n$, the sum $A +_n B$ is the $n \times n$ matrix C and the product $A \times_n B$ is the $n \times n$ matrix D defined by

$$A +_n B \quad =_{df} \quad C, \quad \text{where } c_{ij} = a_{ij} + b_{ij} \quad i, j = 1, \ldots, n\,;$$

$$A \times_n B \quad =_{df} \quad D, \quad \text{where } d_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj} \quad i, j = 1, \ldots, n\,.$$

Matrix Multiplication VII

The following important property is stated as an exercise:

Exercise 1. *Let* $\mathcal{M}_n = (M_n, +_n, \times_n, 0_n, I_n)$. *Then we have:* $\mathcal{M}_n$ *is a ring if and only if* R *is ring.*

Note that we did not require the ring R to be a commutative one in Exercise 1. It should be noted that the matrix multiplication $\times_n$ as defined above, is *not* commutative for $n > 1$, even if the multiplication $\cdot$ in the underlying ring R is commutative. In the following, we often omit the subscript n in $+_n$ and $\times_n$, i.e., we just write $+$ and $\times$, when there is no possibility of confusion. Moreover, we often just write AB instead of $A \times B$ to simplify notation.

Matrix Multiplication VII

The following important property is stated as an exercise:

Exercise 1. *Let* $\mathcal{M}_n = (M_n, +_n, \times_n, 0_n, I_n)$. *Then we have:* $\mathcal{M}_n$ *is a ring if and only if* R *is ring.*

Note that we did not require the ring R to be a commutative one in Exercise 1. It should be noted that the matrix multiplication $\times_n$ as defined above, is *not* commutative for $n > 1$, even if the multiplication $\cdot$ in the underlying ring R is commutative. In the following, we often omit the subscript n in $+_n$ and $\times_n$, i.e., we just write $+$ and $\times$, when there is no possibility of confusion. Moreover, we often just write AB instead of $A \times B$ to simplify notation.

Next, we continue with a very useful technical result. Let R be a commutative ring with 1, and let $\mathcal{M}_n$ be the ring of all $n \times n$ matrices over R.

Matrix Multiplication VIII

Assume n to be even. Then, we can divide any $n \times n$ matrix A into 4 matrices $A_{11},\ A_{12},\ A_{21},\ A_{22}$ of size $n/2 \times n/2$, i.e.,

$$A = \left(\begin{array}{ccc|ccc} a_{11} & \cdots & a_{1,\frac{n}{2}} & a_{1,\frac{n}{2}+1} & \cdots & a_{1n} \\ \vdots & & & & & \vdots \\ a_{\frac{n}{2}1} & \cdots & a_{\frac{n}{2},\frac{n}{2}} & a_{\frac{n}{2},\frac{n}{2}+1} & \cdots & a_{\frac{n}{2}n} \\ \hline a_{\frac{n}{2}+1,1} & \cdots & a_{\frac{n}{2}+1,\frac{n}{2}} & a_{\frac{n}{2}+1,\frac{n}{2}+1} & \cdots & a_{\frac{n}{2}+1,n} \\ \vdots & & & & & \vdots \\ a_{n1} & \cdots & a_{n,\frac{n}{2}} & a_{n,\frac{n}{2}+1} & \cdots & a_{nn} \end{array}\right)$$

$$= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}.$$

Matrix Multiplication IX

Furthermore, let $R_{2,n/2}$ be the ring of all 2×2 matrices with elements from $M_{n/2}$. Then, the multiplication and addition of matrices from M_n is equivalent to the multiplication and addition of the corresponding 2×2 matrices from $R_{2,n/2}$. That is, for $A,\ B \in M_n$ and $A_{11},\ A_{12},\ A_{21},\ A_{22}\ B_{11},\ B_{12},\ B_{21},\ B_{22} \in R_{2,n/2}$ we have

$$A + B = \begin{pmatrix} A_{11} + B_{11} & A_{12} + B_{12} \\ A_{21} + B_{21} & A_{22} + B_{22} \end{pmatrix}; \tag{4}$$

$$A \times B = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}. \tag{5}$$

Matrix Multiplication X

Now, we are ready to deal with the complexity of matrix multiplications. What we are going to count here is the number of the arithmetic ring operations, i.e., additions and multiplications in the underlying ring R. Our first theorem establishes the starting point by analyzing the obvious matrix multiplication algorithm that is based on the definition of matrix multiplication.

Matrix Multiplication X

Now, we are ready to deal with the complexity of matrix multiplications. What we are going to count here is the number of the arithmetic ring operations, i.e., additions and multiplications in the underlying ring R. Our first theorem establishes the starting point by analyzing the obvious matrix multiplication algorithm that is based on the definition of matrix multiplication.

Theorem 4

The usual algorithm for multiplying any two $n \times n$ *matrices requires* $2n^3 - n^2$ *arithmetic operations.*

Matrix Multiplication XI

Proof. Let $A = (a_{ij})$ and let $B = (b_{ij})$, $i, j = 1, \ldots, n$ be any two $n \times n$ matrices. Then the product $C = (c_{ij}) = A \times B$ is given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \, .$$

That is, the computation of every element c_{ij} requires n ring multiplications and $n - 1$ ring additions. Since there are n^2 many elements c_{ij} which have to be computed, the total number of ring multiplications is n^3 and the total number of ring additions is $n^2(n-1)$. Thus the overall number of ring operations is $2n^3 - n^2$. ∎

Matrix Multiplication XII

Question

Can we do any better?

Matrix Multiplication XII

Question

Can we do any better?

The affirmative answer was found by Volker Strassen.

Theorem 5 (Strassen (1969))

There is an algorithm for multiplying any two $n \times n$ *matrices that requires* $O(n^{\log 7})$ *ring operations.*

Matrix Multiplication XIII

Proof. Let $T(n)$ denote the number of ring operations needed for multiplying any two $n \times n$ matrices. Furthermore, let $A, B \in M_n$ be any two $n \times n$ matrices. First, we assume n to be a power of 2. Using Equation (5) we can write

$$\begin{aligned} C &= A \times B \\ &= \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix} \\ &= \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \end{aligned}$$

This approach alone does not help, since we have reduced the original problem of size n into eight subproblems of size $n/2$. Thus, the resulting recursive equation has the solution $T(n) = O(n^{\log 8}) = O(n^3)$.

Matrix Multiplication XIV

At this point Strassen (1969) discovered the following: Let

$$\begin{aligned}
M_1 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\
M_2 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
M_3 &= (A_{11} - A_{21})(B_{11} + B_{12}) \\
M_4 &= (A_{11} + A_{12})B_{22} \\
M_5 &= A_{11}(B_{12} - B_{22} \\
M_6 &= A_{22}(B_{21} - B_{11}) \\
M_7 &= (A_{21} + A_{22})B_{11}\,,
\end{aligned}$$

Matrix Multiplication XV

then an easy calculation shows that

$$\begin{aligned} C_{11} &= M_1 + M_2 - M_4 + M_6 \\ C_{12} &= M_4 + M_5 \\ C_{21} &= M_6 + M_7 \\ C_{22} &= M_2 - M_3 + M_5 - M_7 \, . \end{aligned}$$

Matrix Multiplication XV

then an easy calculation shows that

$$\begin{aligned} C_{11} &= M_1 + M_2 - M_4 + M_6 \\ C_{12} &= M_4 + M_5 \\ C_{21} &= M_6 + M_7 \\ C_{22} &= M_2 - M_3 + M_5 - M_7 \, . \end{aligned}$$

Thus, we have reduced the original problem of size n to *seven* multiplications of matrices having size $n/2 \times n/2$ and 18 additions of $n/2 \times n/2$ matrices.

Matrix Multiplication XVI

Consequently, for $n \geqslant 2$ we directly get the recursive equation

$$T(n) = 7 \cdot T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2,$$

which has the solution $T(n) = 7 \cdot 7^{\log n} - 6n^2$, and hence $T(n) = O(7^{\log n}) = O(n^{\log 7})$.

If n is not a power of 2, then we embed each matrix in a matrix whose dimension is the next-higher power of 2. This at most doubles the dimension and thus increases the constant by at most a factor of 7. Hence, $T(n) = O(n^{\log 7})$ for all $n \in \mathbb{N}^+$. ∎

Matrix Multiplication XVII

Strassen's (1969) matrix multiplication algorithm is also not the best possible. After his pioneering paper, many researchers worked on even faster matrix multiplication algorithms. The currently best known algorithm achieves

$$O(n^{2.376})$$

and is due to Coppersmith and Winograd (1990).

Thank you!

Sir Isaac Newton

Stephen A. Cook

Volker Strassen