MODEL – Arithmetics

October 12, 2017

About the previous course. All the definitions of basic algebraic structures must be known. Any student should be able to prove or disprove that a given set with given binary operations is a group, a ring or a field, or a vector space.

Also, given $n \in \mathbb{N} - \{0\}$, the definition of $\mathbb{Z}/n\mathbb{Z}$ must be known ; running operations over such a set should be routine.

Cryptography, computer vision, and many other areas require routinely to perform operations with integers, polynomials and matrices of *large* size. Current software can routinely perform operations such as:

- multiply integers with more than 30 000 000 digits ;
- multiply univariate polynomials of degree $\simeq$ 650 000 (with coefficients in $\mathbb{Z}/p\mathbb{Z}$ where p is a prime number like 67108879 (less than 2^{26});

Such operations require *fast* arithmetics and care about the *size* of the data which are manipulated during these computations.

Complexity models. We mainly focus on time complexity ; space complexity issues will be mostly ignored in this course.

We will use the Random Access Machine model which is the most standard one. In this model, a program reads and writes integers on different tapes, it is also allowed to use an arbitrary number of integer registers. Elementary operations (which coincide with those supported by an assembly code) are

- write and read (on a tape or a register) ;
- addition, substraction, multiplication and division ;
- three jump instructions: unconditional jump, jump when a register is 0 or jump when a register is not 0.

We won't use subtelties of this model in what follows ; the students are encouraged to read a little bit more about this model in the academic litterature.

We will measure two kinds of complexities of algorithms

- *Arithmetic complexity.* Most of algorithms running on mathematical data are expected to run on an abstract algebraic structure equipped with binary operations.

  In this model, we only count the number of such binary operations and predicate tests performed by the algorithm.

  This model reflects what is observed in practice when the cost of running binary operations is mostly a constant.

  For most of algorithms running over $\mathbb{Q}$, such a model is not so convenient when the size of manipulated numbers grows significantly during the computations.

  Note that in this model, we do not take into account copy operations and countings performed for running loops, etc.

- *Bit complexity.* This model is devoted to take into account the growth of numbers when running algorithms. It is primarily defined for $\mathbb{Z}$: we will see below that every integer can be seen as a *vector* of elements of $\{0, \ldots, b-1\}$ for a given integer $b \in \mathbb{N}$. In the bit complexity model, we count the number of operations performed in $\mathbb{Z}/b\mathbb{Z}$.

  Again memory access is neglicted in this model (for instance, transposing a matrix, even an extremely large one, is free in this model).

We will use the $O(.)$ notation ; in short we say that $f(n) = O(g(n))$ when there exists $K > 0$ and $M > 0$ such that for all $n \geq M$, the following inequality holds:

$$\|f(n)\| \leq K\|g(n)\|$$

Careful attention must be paid when several parameters are involved in complexity estimates. In those cases, without further precisions, all of them are expected to tend to infinity and the constant K should not depend on any of these parameters.

We may also use the $O\tilde{}(.)$ notation: we say that $f(n) = O\tilde{}(g(n))$ when there exists $a \geq 0$ such that

$$f(n) = O(g(n) \log_2^a(\max(2, \|g(n)\|)).$$

Definition 1. *Let N be the sum of the sizes of the input and output data. An algorithm is said to be nearly optimal (or quasi optimal) when its runtime is bounded by $O\tilde{}(N)$.*

Integers. We present here how integers are encoded in computers. A key requirement is unicity (canonicity) of course and generality.

Unicity of integer encodings presented below is actually a consequence of the unicity of quotient and remainder in the Euclidean division of integers.

Proposition 2. *Let a and b be in $\mathbb{N}$. There exist unique (q, r) in $\mathbb{N} \times \mathbb{N}$ such that $a = bq + r$ and $0 \leq r < b$.*

Proof. Assume by contradiction that there exist distinct couples (q, r) and (q', r') such that $0 \leq r < b$ and $0 \leq r' < b$ and $a = bq + r = bq' + r'$. One may also assume that $q \neq q'$ (else we would immediately have $a - bq = a - bq'$ which implies $r = r'$, a contradiction). Hence, one can assume without loss of generality $q > q'$.

Then the following holds:

$$b(q - q') = r' - r.$$

Since $q > q'$, we deduce that $r' - r > 0$. Since $r' < b$ and $r > 0$, we deduce that $r' - r$ is a positive integer which is less than b ($r < b$). Besides, $b(q - q')$ is an integer which is greater than 1 (> 1), hence a contradiction. □

Theorem 3. *Let $n \in \mathbb{N}$ and $b \in \mathbb{N} - \{0\}$. There exists a unique sequence $a_0, \ldots, a_h$ in $\mathbb{Z}/b\mathbb{Z}$ such that $a_h \neq 0$*

$$a_0 + a_1.b^1 + \cdots + a_h b^h = n.$$

Such a decomposition is called decomposition of n in the basis b. Besides, h is called the height of n in base b.

Proof. Our proof is by induction on n. The case $n = 1$ is obvious. Our induction assumption is that for any $m < n$, the decomposition of m in basis b exists and is unique.

By definition, a_0 is the remainder of the Euclidean division of n by b (and $q = a_1 + \cdots + a_h b^h$ is the unique quotient of such a division). By Proposition 2, a_0 is unique (and the aforementioned quotient as well).

Observe that $q < n$. Hence applying our induction assumption on q finishes the proof. □

Lemma 4. *Let $n \in \mathbb{N} - \{0\}$ and $b \in \mathbb{N}$. Then the height of n in basis b is bounded by $\lceil \log_b(n) \rceil$.*

Proof. We obviously have $b^h \leq n < b^{h+1}$. □

Basic exercise. How many bits do you need to store integers of the form a^t for $a \in \mathbb{N}$ and $t \in \mathbb{N}$?

What is the bit complexity of the basic algorithm computing the n-th term of the Fibonacci sequence ? [recall that it is defined by $F_{k+2} = F_{k+1} + F_k$ with $F_0 = F_1 = 1$].

Bit complexity counts the number of single / bit operations and hence allows to take into account the growth of coefficients.

Theorem 5. *Let a and b be integers of respective bit size h_1 and h_2 ; we also let $h = \max(h_1, h_2)$. There exist algorithms for*

- *adding a and b in bit complexity $O(\max(h_1, h_2))$;*
- *multiplying a and b in bit complexity $O(h_1 h_2)$ (naive algorithm) ;*
- *multiplying a and b in bit complexity $O(h^{1.59})$ (Karatsuba's algorithm) ;*
- *multiplying a and b in bit complexity $O(h \log(h) \log(\log(h)))$ (Fast Fourier Transform).*

In the sequel, the notation $\mathsf{M}_{\mathbb{Z}}(h)$ stands for the bit complexity of multiplying two integers of bit size $\leq h$.

Basic exercise. Let $n \in \mathbb{N}$. What is the (binary) cost of performing addition and multiplication on $\mathbb{Z}/(n\mathbb{Z})$?

Would you implement the same way the arithmetic in $\mathbb{Z}/(n\mathbb{Z})$ when $n < 2^{32}$ or $n > 2^{32}$?

Polynomials. Let R be a ring. Univariate polynomials in $R[X]$ such as $c_0 + c_1 X + \cdots + c_d X^d$ can be represented by vectors of coefficients $[c_0, \ldots, c_d]$. Hence there is a strong similarity with integers (the variable X in some sense plays the same role as the basis b chosen to represent integers).

On noticeable fact is about *sparse* polynomials: these are polynomials such that the number of their non-zero coefficients is *negligeible* compared to their degree. Indeed, remark that the proposed encoding as vectors of coefficients may be too costly for a polynomial such as $X^{425147} + 7\, X^{26245} - 11\, X^{24} + 3$.

Theorem 6. *Let f and g be two polynomials in $R[X]$ of degree bounded by n. Then the arithmetic complexity of multiplying f and g requires*

- *$O(n^2)$ operations in R using the naive algorithm ;*
- *$O(n^{1.59})$ operations in R using Karatsuba's algorithm ;*
- *can be performed within $O(n \log(n) \log(\log(n)))$ arithmetic operations (Fast Fourier Transform).*

The proof of the first statement is obvious. Students are expected to try to make it *and to come at next course with a proof of that statement to be given to the teacher (that will count for your final grade).*

We take now advantage of the mention of Karatsuba's algorithm to study the famous master theorem.

First let us recall that Karatsuba's algorithm relies on the very basic following observation: writing

$$f = f_0 + f_1\, X \qquad \text{and} \qquad g = g_0 + g_1\, X$$

their product $h = fg = h_0 + h_1\, X + h_2\, X^2$ can be obtained through an interpolation formula at 0, 1 and ∞:

$$h_0 = f_0 g_0, \quad h_2 = f_1 g_1 \quad h_1 = f(1)g(1) - h_0 - h_2.$$

Hence, to obtain the coefficients of h, only 3 multiplications (and 4 additions) are needed.

The idea of reducing the number of multiplications to speed up foundational algorithms will be used repeatedly this year.

One derives a multiplication algorithm by truncating polynomials at their half degree. This is a famous example of a divide and conquer strategy which is at the foundations of many asymptotically fast algorithms.

On input data of size n, this principle consists in reducing to m recursive calls with input data divided by p (usually $p = 2$) and finally recombine the result. The cost of the splitting and recombination steps is bounded by a function T. When the input data have a size smaller than s, an other algorithm running in time κ is called.

We describe below how the runtime of a divide-and-conquer algorithm can be estimated. The total cost can be written as

$$C(n) = \begin{cases} T(n) + mC(\lceil n/p \rceil) & \text{when } n \geq s \geq p \\ \kappa & \text{else} \end{cases} \tag{1}$$

Lemma 7 (Divide and conquer lemma)**.** *Let C be a function as in (1) with $m > 0, \kappa > 0$ and T such that $T(pn) \geq qT(n)$ with $q > 1$. Then, if n is a positive power of p, one has*

$$C(n) \leq \begin{cases} \left(1 - \frac{m}{q}\right)^{-1} T(n) + \kappa n^{\log_p(m)} & \text{if } q > m \\ T(n)\log_p(n) + \kappa n^{\log_p(q)} & \text{if } q = m \\ n^{\log_p(m)} \left(\frac{T(n)}{n^{\log_p(q)}} \frac{q}{m-q} + \kappa\right) & \text{if } q < m. \end{cases}$$

Proof. The repeated use of (1) yields

$$\begin{aligned} C(n) \leq & \quad T(n) + mC\left(\frac{n}{p}\right) \\ \leq & \quad T(n) + mT\left(\frac{n}{p}\right) + \cdots + m^{k-1}T\left(\frac{n}{p^{k-1}}\right) + m^k C\left(\frac{n}{p^k}\right) \\ \leq & \quad T(n)\left(1 + \frac{m}{q} + \cdots + \left(\frac{m}{q}\right)^{k-1}\right) + m^k \kappa \end{aligned}$$

for $k = \lfloor \log_p(n/s) \rfloor + 1$. This choice implies

$$m^k \kappa \leq n^{\log_p(m)} \kappa$$

which bounds the second term of the above inequality. It remains to bound the first term.

When $m < q$, the sum (between parentheses) is routinely bounded by the geometric series.

When $m = q$, this is a sum of equalled terms.

When $m > q$, it can be rewritten as

$$\left(\frac{m}{q}\right)^{k-1} \left(1 + \frac{q}{m} + \cdots + \left(\frac{q}{m}\right)^{k-1}\right)$$

which we also bound routinely. □

On easily deduces from that lemma the following master theorem.

Theorem 8 (Master theorem)**.** *Let C be a function as in (1) with $m > 0, \kappa > 0$ and T be an* increasing *function such that there exist q and r with $1 < q \leq r$ sastisfying*

$$qT(n) \leq T(pn) \leq rT(n), \qquad \text{for all } n \text{ large enough.}$$

Then

$$C(n) = \begin{cases} O(T(n)) & \text{if } q > m, \\ O(T(n)\log(n)) & \text{if } q = m, \\ O(n^{\log_p(m)} \frac{T(n)}{n^{\log_p(q)}} & \text{if } q < m. \end{cases}$$

The master formula for Karatsuba's algorithm is

$$K(n) \leq 3K(n/2) + 4n$$

We state now our divide and conquer lemma and master theorem.

Basic exercise. Can we deduce from the above theorem the bit complexity of multiplying polynomials i8n $\mathbb{Z}[X]$?

In the rest of the course, we will assume that polynomials are encoded with vectors of coefficients.

Matrices. Matrices will be mostly considered with entries in a field $\mathbb{K}$ (either a finite one like $\mathbb{Z}/p\mathbb{Z}$ with p prime or a field of characteristic 0). Dense matrices are encoded with an array ; sparse and structured matrices are encoded with ad-hoc representations minimizing space complexity.

Adding matrices is easy.

Lemma 9. *Let M and N be $p \times q$ matrices with entries in $\mathbb{K}$. Computing $M + N$ is done using $O(p + q)$ operations in $\mathbb{K}$.*

Proof. The proof is left to the students ; that is really very easy. □

Multiplying matrices is more difficult.

Lemma 10. *Let M and N be $n \times n$ matrices with entries in $\mathbb{K}$. Computing the product $M.N$ with the naive multiplication algorithm uses $O(n^3)$ operations in $\mathbb{K}$.*

Remark and warning. The size of the entries M, N is $O(n^2)$; the size of the output is $O(n^2)$. We deduce that the cost of multiplying M by N is *not* asymptotically optimal. **Never** write and/or say that matrix multiplication is quadratic (unless it is proved in the future which is not likely) ; you will be ridiculous if you do that.

Hence, the quest for nearly optimal algorithms for matrix multiplication is not finished.

It is actually a difficult problem. The first algorithm with a time complexity better than $O(n^3)$ is due to Strassen. As for Karatsuba's algorithm, it is based on an improvement in the case $n = 2$ (for Karatsuba, we were considering quadratic polynomials to built on the general improvement) and a decrase of the number of multiplications.

So, let

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \text{and} \qquad M = \begin{bmatrix} x & y \\ z & t \end{bmatrix}.$$

To multiply M by N, compute the following quantities

$$\begin{array}{lll} q_1 = a(x+z) & q_2 = d(y+t) & \\ q_3 = (d-a)(z-y) & q_4 = (b-d)(z+t) & \\ q_5 = (b-a)z & q_6 = (c-a)(x+y) & q_2 + q_7 \end{array},$$

compute the sums

$$\begin{array}{ll} r_{1,1} = q_1 + q_5 & r_{1,2} = q_2 + q_3 + q_4 - q_5 \\ r_{2,1} = q_1 + q_3 + q_6 - q_7 & r_{2,2} = q_2 + q_7 \end{array}$$

The entry at row i and column j of the result is exactly $r_{i,j}$ (for $1 \leq i, j \leq 2$).

Hence, this remark allows to compute matrix multiplication by splitting $2n \times 2n$ matrices into blocks of size n. It performs 7 recursive calls to the matrix multiplication algorithm and $C = 18$ additions.

The recurrence formula for estimating the cost of multiplying square matrices is

$$S(n) \leq 7S\left(\frac{n}{2}\right) + C\left(\frac{n}{2}\right)^2.$$

Using the above lemma for estimating the cost of divide and conquer strategies with $m = 7$, $p = s = 2$, $\kappa = 1$ and $q = 4$, we deduce

$$S(n) \leq \left(1 + \frac{C}{3}\right) n^{\log_2(7)}.$$

This leads to the following theorem (using $\log_2(7) \leq 2.81$).

Theorem 11. *One can multiply $n \times n$ matrices with entries in a ring R using $O(n^{2.81})$ operations in R.*

Strassen's algorithm becomes competitive for matrices of size $\simeq 64$ with entries which are integers of small size.