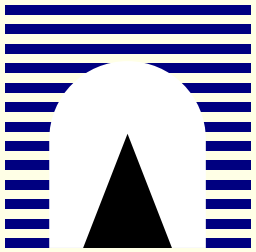

Basic Algorithms in Number Theory

Francesco Pappalardi

#1 - Algorithmic Complexity & more.

August 31ST 2015

SEAMS School 2015
Number Theory and Applications in Cryptography and Coding Theory
University of Science, Ho Chi Minh, Vietnam
August 31 - September 08, 2015

What is an algorithm and what is its *complexity*?

- An *algorithm* takes `Inputs` and produces `Outputs`

- The *Complexity* (or *running time*) of an algorithm A is a function $C_A(n) = \max\{\text{cost of running } A \text{ in } I \mid I \text{ is an input of size} \leq n\}$.
- The *cost of running* depends on the context.
  It is measured in terms of the number of *elementary operations* that the algorithm performs.

- The *input size* also depends on the context
  (many times we will use the number of digits)

- All these concepts can be formalized. However, we adopt a naive approach and we immediately specialize to the number theoretic set up.

What is the size of an integer?

If $x \in \mathbb{Z}, x \neq 0$, the $\text{size}_b(x)$ is the number of digits of x in base b. That is

$$\text{size}_b(x) := 1 + \lfloor \log_b(|x|) \rfloor$$

where $\log_b$ denotes the logarithm in base b and $\lfloor u \rfloor$ is the floor of u (i.e. the largest integer smaller than or equal to u.

We have that $\text{size}_b(x) = O(\log |x|)$.

We write that $g(x) = O(f(x))$ if there exists $C > 0$ such that $|g(x)| \leq C|f(x)|$ for all sufficiently large x.

Note that if $a, b > 1$ are fixed, then $\log_a(|x|) = O(\log_b(|x|)$. Therefore when using the O–notation the choice of b is irrelevant.

We use the O–notation to estimate the complexity of Algorithms. We say that an algorithm runs in polynomial time if its complexity on inputs of size up to n, is $O(n^k)$ for some $k > 0$.

PROBLEM 1. MULTIPLICATION: for $x, y \in \mathbb{Z}$, find $x \cdot y$.

- **School Multiplication Algorithm:** It requires about n^2 digit-sized multiplications followed by n sums of integers of size about n.
  - Since to add two n–sized integers, about n digit-sized operations are necessary,
  - The complexity to multiply two n-sized integers using the School Multiplication Algorithm is $O(n^2) + nO(n) = O(n^2)$.
- **Karatsuba Multiplication Algorithm (1960):**
  It uses multiplication of polynomials

$$(a + bX)(c + dX) = ac + (ad + bc)X + bdX^2 = ac + ((a + b)(c + d) - ac - bd)X + bdX^2$$

  It has complexity $O(n^{\log_2 3})$.
- **Schönhage Multiplication (1971):** It has complexity $O(n \log n \log\log n)$ on n–digit number (algorithms that use it are said to use *fast arithmetics*;*(sometimes we write* $O(n^{1+\varepsilon})$*).*

PROBLEM 2. EXPONENTIATION: for $x \in \mathbb{Z}$ and $n \in \mathbb{N}$, find x^n.

Here we assume that x is fixed and we review algorithms whose complexity depends on the size of n. (It is easy to check that the complexity of exponentiation is $O(n)$).

Example: To compute x^{16} are clearly enough 15 multiplications. However since

$$x^{16} = \left(\left(\left(x^2\right)^2\right)^2\right)^2,$$

only 4 squaring are enough!!

The binary expansion of n has a role in efficient exponentiation.

If $n = \sum a_i 2^i$ with $a_i \in 0, 1$, then

$$x^n = x^{a_0}(x^2)^{a_1}(x^4)^{a_2} \dots .$$

The idea also works when x is the element of any multiplicative group (or a monoid).

Right-to-Left Exponentiation

Input: x in a fixed group and $n \in \mathbb{N}$
Output: x^n
1. $y := 1$
2. While $n > 0$,
   if n is odd $y := x \cdot y$
   $x := x^2, n := \lfloor n/2 \rfloor$
3. Return y

where the *floor* $\lfloor u \rfloor$ of u denotes the largest integer less than or equal to u.

The proof is by induction and gives the recursive algorithm

$$\mathrm{Exp}(x,n) = \begin{cases} 1 & \text{if } n = 0, \\ \mathrm{Exp}(x^2, n/2) & \text{if } n > 0 \text{ is even}, \\ x\,\mathrm{Exp}(x^2, (n-1)/2) & \text{if } n \text{ is odd}. \end{cases}$$

Complexity is $O(\log n)$. Very important applications in Number Theory.

Left-to-Right Exponentiation

Using the mathematical equivalence of algorithms:

$$\mathrm{Exp}(x,n) = \begin{cases} 1 & \text{if } n = 0, \\ \mathrm{Exp}(x,n/2)^2 & \text{if } n > 0 \text{ is even}, \\ x\,\mathrm{Exp}(x,(n-1)/2)^2 & \text{if } n \text{ is odd}. \end{cases}$$

and unfolding it into an iterative algorithms:

Input: x in a fixed group, $n \in \mathbb{N}$ and $m = 2^a$ with $m/2 \le n < m$

Output: x^n

1. $y := 1$
2. While $m > 1$,
   $m := m/2, y := y^2$
   if $n \ge m$ $y := x \cdot y$, $n := n - m$
3. Return y

The ring $\mathbb{Z}/m\mathbb{Z}$ $(m > 1)$

The cost of computing x^n is $O(\log n)$ if the cost of multiplication in the monoid G is bounded.

A very important case is when $G = (\mathbb{Z}/m\mathbb{Z})^*$.

The ring $\mathbb{Z}/m\mathbb{Z}$ is the ring whose elements are the arithmetic progressions modulo m.

We know that $\mathbb{Z}/m\mathbb{Z}$ has m elements, namely $k + m\mathbb{Z}$ where $k = 0, 1, \ldots, m-1$.

Sometimes we abuse the notation and write $\mathbb{Z}/m\mathbb{Z} = \{0, 1, \ldots, m-1\}$. With this abused notation we have, for $a, b \in \mathbb{Z}/m\mathbb{Z}$

$$a +_m b := \begin{cases} a + b & \text{if } a + b < m \\ a + b - m & \text{otherwise} \end{cases} \quad \text{and} \quad a \times_m b := a \cdot b \mod m.$$

The ring $\mathbb{Z}/m\mathbb{Z}$ continues

$$a +_m b := \begin{cases} a+b & \text{if } a+b < m \\ a+b-m & \text{otherwise} \end{cases} \quad \text{and} \quad a \times_m b := a \cdot b \mod m$$

The symbol $u \mod m$ denoted the *remainder* of the division of u by m. That is the unique integer r such that

1. $0 \leq r < m$,

2. $u = qm + r$ for some $q \in \mathbb{Z}$.

It can be shown that, if $u, m \in \mathbb{Z}$, $m > 1$ then $u = qm + r$ can be computed in time $O((\log m)(\log q)) = O(\log^2 \max(|u|, m))$ with naive algorithms and in time $O(\log^{1+\epsilon} \max(|u|, m))$ using fast arithmetics.

The ring $\mathbb{Z}/m\mathbb{Z}$ continues

CONSEQUENCE: Operations in $\mathbb{Z}/m\mathbb{Z}$ can be performed in time

	(scholarly)	(fast arithmetics)
addition	$O(\log m)$	
multiplication	$O(\log^2 m)$	$O(\log^{1+\epsilon} m)$
exponentiation by n	$O(\log n \log^2 m)$	$O(\log n \log^{1+\epsilon} m)$
inverses	$O(\log^2 m)$	$O(\log^{1+\epsilon} m)$

NOTE. There is also an efficient old method to compute the inverses in

$$(\mathbb{Z}/m\mathbb{Z})^* = \{a \in \mathbb{Z}/m\mathbb{Z} \text{ such that there exists } b \text{ with } ab \equiv 1 \bmod n\}.$$

This will be one of the highlights of tomorrow's lecture.

PROBLEM 3. GCD: Given $a, b \in \mathbb{N}$ find $\gcd(a, b)$

The non negative $\gcd(a, b)$ is the greatest common divisor of a and b. Note that

$$\gcd(a, 0) = a \quad \text{and} \quad \gcd(a, b) = \gcd(b, a \bmod b).$$

This observation leads to the algorithm:

```
Input:   a, b ∈ ℕ
Output:  gcd(a, b)
While b > 0,
        {a, b} := {b, a mod b}
Return a
```

Since the number of times the loop is iterated in $O(\log \max\{a, b\})$, the complexity of this algorithm is certainly $O(k^3)$ on k-bits integers but we will do much better tomorrow.

PROBLEM 4. PRIMALITY: Given $n \in \mathbb{N}$ odd, determine if it is prime

This is our first example of *decision problem*, for which the Output is "*yes*" or "*no*".

It is easy to check if a number is prime with *trial division*. The complexity of such an algorithm is $O(\sqrt{n})$ which is exponential.

Fermat Little Theorem. *If n is prime and $a \in (\mathbb{Z}/n\mathbb{Z})^*$, then the multiplicative order of a divides $n-1$ (i.e. $a^{n-1} \equiv 1 \bmod n$).*

Note that FTL can be checked on n in time $O(\log^3 n)$ so it provides (often) a good way to check that a number is composite.

Example: $2^{1000} \bmod 1001 = 562$ implies that 1001 is not prime and we haven't even tried to factor it

Primality continues

However from the idea of FLT we deduce a primality test:

Theorem. *If n is an integer and $a \in (\mathbb{Z}/n\mathbb{Z})^*$ such that $a^{n-1} \equiv 1 \bmod n$, and $a^{(n-1)/q} \not\equiv 1 \bmod n$ for all prime divisors q of $n-1$, then n is prime.*

Proof. The statement is just rephrasing of the fact that $(\mathbb{Z}/n\mathbb{Z})^*$ is cyclic (generated by a) and has order $n-1$.

Since $\#(\mathbb{Z}/n\mathbb{Z})^* = \varphi(n)$ (the Euler function), the conclusion follows from the fact the $\varphi(n) = n-1$ iff n is prime. □

Note: FLT is of any use to determine primality only if we can factor $n-1$. For example it can be shown that $n = 15 \times 2^{1518} + 1$ is prime since $11^{n-1} \bmod n = 1$ and

$11^{\frac{n-1}{2}} \bmod n = 137919\cdots, 11^{\frac{n-1}{3}} \bmod n = 79851\cdots$ and $11^{\frac{n-1}{5}} \bmod n = 134287\cdots$

However it is seldom the case that $n-1$ can be factored.

Primality continues

A "more" useful result:

Theorem (Pocklington). *If n is an integer, $a \in (\mathbb{Z}/n\mathbb{Z})^*$ and m is a divisor of $n-1$ with $m > \sqrt{n}$ such that $a^{n-1} \equiv 1 \bmod n$, and $\gcd(a^{(n-1)/q} - 1, n) = 1$ for all prime divisors q of m, then n is prime.*

Therefore *proving Primality is easy if $n-1$ can be "half factored"*

Assuming that $a^{n-1} \bmod n$ is a random integer modulo n, one could think that the above idea could be pushed further. However there are composite number, called *Carmichael numbers* such that

$$a^{n-1} \equiv 1 \bmod n \quad \forall a \in (\mathbb{Z}/n\mathbb{Z})^*.$$

For example $561 = 3 \times 11 \times 17$ is the smallest Carmichael number.

Theorem (AGP).(Alford, Granville, Pomerance (1994)) *There are infinitely many Carmichael numbers.*

If p is an odd prime, then $a^{(p-1)/2} \equiv \pm 1 \mod p$ since $\mathbb{Z}/p\mathbb{Z}$ is a field!!

Legendre Symbols

We take the "square root" of the Fermat congruence.

Definition. $a \in \mathbb{Z}$ is a quadratic residue modulo a prime p if $\exists x \in \mathbb{Z}$ not divisible by p such that $a \equiv x^2 \bmod p$. (i.e. a is a square modulo p).
Furthermore the Legendre symbol

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } p \text{ divides } a, \\ 1 & \text{if } a \text{ is a quadratic residue modulo } p, \\ -1 & \text{if } a \text{ is a quadratic nonresidue modulo } p. \end{cases}$$

Theorem (Euler). $\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \bmod p$.

The above can be checked (scholarly) in time $O(\log^3 p)$

We want to extend the definition of $\left(\frac{a}{p}\right)$ to the case when p non necessarily prime but still odd.

Jacobi Symbols

Definition. Let $a, b \in \mathbb{Z}$ with $b > 1$ odd. The Jacobi symbol is defined as

$$\left(\frac{a}{b}\right) = \prod_{p^{\alpha_p} \| b} \left(\frac{a}{p}\right)^{\alpha_p}.$$

Properties of the Jacobi Symbols:

1. if b is prime, the Jacobi symbols and Legendre's symbols are the same,
2. Jacobi symbols are multiplicative in the numerators and denominators,
3. $\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right)$; so that $\left(\frac{a}{b}\right) = 0$ iff $\gcd(a, b) \neq 1$,
4. (Quadratic Reciprocity) $\left(\frac{a}{b}\right)\left(\frac{b}{a}\right) = (-1)^{(a-1)(b-1)/4}$
5. $\left(\frac{-1}{b}\right) = (-1)^{(b-1)/2}$, $\left(\frac{2}{b}\right) = (-1)^{(b^2-1)/8}$

From the above we can extract an algorithm to compute the Jacobi symbol without factoring the denominator!

Computation of Jacobi (and Legendre) symbols

Input: $a \in \mathbb{Z}, b \in \mathbb{N}$ odd

Output: $\left(\frac{a}{b}\right) \in \{0, 1, -1\}$

$(X, Y, Z) := (a, b, 1)$;

1. if $X = 0$ or $X = 1$, Return $X \cdot Z$,
2. if $X < 0$, $(X, Y, Z) = (-X, Y, Z \cdot (-1)^{(Y-1)/2})$,
3. if $X \geq Y$, $(X, Y, Z) = (X \bmod Y, Y, Z)$,
4. if X is even, $(X, Y, Z) = (X/2, Y, Z \cdot (-1)^{(Y^2-1)/8})$,
5. if X is odd $(X, Y, Z) = (Y \bmod X, X, Z \cdot (-1)^{(X-1)(Y-1)/4})$,
6. Goto 1

The algorithm, incidentally, also checks if $\gcd(a, b) = 1$ (i.e. if a and b are coprime)

It requires only: *reductions, division by* 2, *sign changes.* Its complexity is really equivalent to the complexity of the gcd-algorithm (with some bookkeeping) which is $O(k^3)$ (scholarly) on integers of size $\leq k$.

Primality continues

We want to apply Jacobi symbols to primality.

Fact: The Fermat number $F_k := 2^{2^k} + 1$ is prime iff $3^{2^{2^k-1}} \equiv -1 \bmod F_k$.

Definition. *A probabilistic (or randomized) algorithm is an algorithm where a "coin flip" is allowed (typically a the cost of* 1 *unit of running time) and it makes the next move depending on the result. The probability of correctness of a probabilistic algorithm is the proportion of the possible inputs of the algorithm for which it provides the correct answer.*

This definition may look vague. However randomized algorithm are ubiquitous in Number Theory. They are so at the level that we will refer to deterministic algorithm as those that are NOT probabilistic.

It is best to review a famous example.

Solovay–Strassen Primality Test

Input: $k \in \mathbb{N}^{>1}$ and $n \in \mathbb{N}$ odd (to be tested)

Output: ‘‘Prime’’ or ‘‘Composite’’

1. For $i = 1, \ldots, k$
   Choose a randomly from $\mathbb{Z}/n\mathbb{Z} = \{0, 1, \cdots, n-1\}$
   if $\gcd(a, n) \neq 1$ then
      Output ‘‘Composite’’ and halt.
   if $\left(\frac{a}{n}\right) \not\equiv a^{(n-1)/2} \bmod n$ then
      Output ‘‘Composite’’ and halt.
2. Output ‘‘Prime’’.

Theorem. *The Solovay–Strassen algorithm returns returns “Composite” if n is composite, “Prime” if n is prime and returns “Prime” with probability $\leq 2^{-k}$ if n is composite. It has polynomial complexity in k and $\log n$.*

Here we will say nothing more about primality. We will say nothing at all about random numbers.

Famous quotation!!!

Un phénomène dont la probabilité est 10^{-50} *ne se produira donc jamais, au moins ne sarait jamais observé.*

- Émil Borel (Les probabilités et sa vie)

PROBLEM 5. Quadratic Nonresidues:

It illustrates why a deterministic life is sometimes unreasonable.

PROBLEM 5. Quadratic Nonresidues: *Given an odd prime p, find a quadratic non residue mod p.*

If p is prime then $\#\{a \,|1 \leq a < p, a \text{ is a quadratic residue } \} = \dfrac{p-1}{2}$ and the same for nonresidues.

Any a has 50% chance of being a quadratic nonresidue

Checking for quadratic residuosity if quite cheap (i.e. fast) via Jacobi symbols.

However no deterministic polynomial time algorithm is known. Nothing seems better then testing $a = 2, 3, 5, 6, 7, 8 \ldots$ until arriving to a nonresidue.

- best known result: least quadratic nonresidue is $O(p^{1/4})$;
- believed that: least quadratic nonresidue is $O(\log^{1+\epsilon} p)$;
- (Bach 1990): ERH implies least quadratic nonresidue is $\leq 2 \log^2 p$.

PROBLEM 6. POWER TEST: Given $n \in \mathbb{N}$, determine if $n = b^k (\exists k > 1)$

This is a mandatory preliminary check in several algorithms

- If $n = b^k$, then $1 < k \leq \log_2 n$;
- for each $k = 2, \ldots, \lfloor \log_2 n \rfloor$, Newton's method for finding roots can be applied to $x^k - n$ so to check if there is an integer root.
- here we will assume that it is doable in polynomial time.

PROBLEM 7. FACTORING: Given $n \in \mathbb{N}$, find a proper divisor of n

- A very old problem and a difficult one;
- Trial division requires $O(\sqrt{n})$ division which is an exponential time (i.e. impractical)
- Several different algorithms
- A very important one uses elliptic curves...
  see next week course by J. Jimenez Urroz.
- we review the elegant Pollard ρ method.

Suppose n is not a power and consider the function:

$$f : \mathbb{Z}/n\mathbb{Z} \longrightarrow \mathbb{Z}/n\mathbb{Z}, \quad x \mapsto f(x) = x^2 + 1.$$

The k-th iterate of f is $f^k(x) = f^{k-1}(f(x))$ with $f^1(x) = f(x)$.

If $x_0 \in \mathbb{Z}/n\mathbb{Z}$ is chosen "sufficiently randomly", the sequence $\{f^k(x_0)\}$ behaves as a random sequence of elements of $\mathbb{Z}/n\mathbb{Z}$ and we exploit this fact.

Pollard ρ factoring method

Input: $n \in \mathbb{N}$ odd and not a perfect power (to be factored)

Output: a non trivial factor of n

1. Choose at random $x \in \mathbb{Z}/n\mathbb{Z} = \{0, 1, \ldots, n-1\}$
2. For $i = 1, 2 \ldots$.

   $g := \gcd(f^i(x) - f^{2i}(x), n)$

   If $g = 1$, goto next i

   If $1 < g < n$ then output g and halt

   If $g = n$ then go to Step 1 and choose another x.

What is going on here?

Is is obviously a probabilistic algorithm but it is not even clear that it will ever terminate.

But in fact it terminates with complexity $O(\sqrt[4]{n})$ which is attained in the worst case (i.e. when n is an RSA module (for RSA see course in Cryptography by K. Chakraborty).

The birthday paradox

Elementary Probability Question: *what is the chance that in a sequence of k elements (where repetitions are allowed) from a set of n elements, there is a repetition?*

Answer: The chance is $1 - \frac{n!}{n^k(n-k)!} \approx 1 - e^{-k(k-1)/2n}$

In a party of 23 friends there 50.04% chances that 2 have the same birthday!!

Relevance to the ρ-Factoring method:

If d is a divisor of n, then in $O(\sqrt{d}) = O(\sqrt[4]{n})$ steps there is a high chance that in the sequence $\{f^k(x_0) \bmod d\}$ there is a repetition modulo d.

Remark (WHY ρ). If $y_1, \ldots, y_m, y_{m+1}, \ldots, y_{m+k} = y_m, y_{m+k+1} = y_{m+1}, \ldots$ and i is the smallest multiple of k with $i \geq m$, then $y_i = y_{2i}$ (the Floyd's cycle trick).

Contemporary Factoring

Contemporary records in factoring are obtained by the Number Field Sieve (NFS) which is an evolution of the Quadratic Sieve (QS). These (together with the ECM-factoring) have sub-exponential heuristic complexity.

More precisely let:

$$L_n[a; c] = \exp\left(((c + o(1)(\log n)^a (\log\log n)^{1-a})\right).$$

which is a quantity that oscillates between exponential $(a = 1)$ and polynomial $(a = 0)$ as a function of $\log n$. Then the complexities are respectively

ECM algorithm with heuristic complexity $L_n[1/2, 1]$ (Lenstra 1987)

NFS algorithm with heuristic complexity $L_n[1/3; 4/3^{3/2}]$ (Pollard)

QS algorithm with heuristic complexity $L_n[1/2, 1]$ (Dickson, Pomerance)

References

[1] J. Buhler & S. Wagon *Basic algorithms in number theory* Algorithmic Number Theory, MSRI Publications Volume **44**, 2008
http://www.msri.org/communications/books/Book44/files/02buhler.pdf

[2] C. Pomerance *Smooth numbers and the quadratic sieve* Algorithmic Number Theory, MSRI Publications Volume **44**, 2008
http://www.msri.org/communications/books/Book44/files/03carl.pdf

[3] R. Crandall and C. Pomerance, *Prime numbers*, 2nd ed., Springer-Verlag, New York, 2005.

[4] E. Bach and J. Shallit, *Algorithmic number theory, I: Efficient algorithms*, MIT Press, Cambridge, MA, 1996.

[5] J. von zur Gathen and J. Gerhard, *Modern computer algebra*, 2nd ed., Cambridge University Press, Cambridge, 2003.

[6] V. Shoup, *A computational introduction to number theory and algebra*, Cambridge University Press, Cambridge, 2005.

[7] These notes http://www.mat.uniroma3.it/users/pappa/missions/slides/ERBIL_1.pdf