Cryptography: notes

Niccolò Simonato

October 11, 2022

Contents

Chapter 1

Elementary operations

1.1 Notation

- Let b be a numeric base.
- Let n be a number in N.
- Length of a number: $l_b(n)$, k. It's equal to $log(n)$.
- (a, b) is the Maximum Common Divisor of a, b.
- Let $n \in N$: $n = (d_{k-1}, d_{k-2}, \ldots, d_1, d_0)$[1].
- $\phi(n)$: the number of elements a in $[1, n]$ such that $(a, n) = 1$.
- $\equiv_p$ is the equivalence in base p. Ex.: $5 \equiv_3 = 5 mod 3 = 2$

1.2 Classification of the algorithms' complexity

In order to better identify the classes of complexity of the algorithms, the following 3 classes are defined:

- Polynomial time: $O(log^{\alpha}(n))$ bit operations, where $\alpha > 0$.
- Exponential time: $O(exp(c \cdot log(n)))$ bit operations, where $c > 0$.
- Sub-exponential time: $O(exp(c \cdot log(n))^{\alpha})$ bit operations, where $c > 0, \alpha \in]0, 1[$.
-

[1] $d_{k-1} \neq 0$

1.3 Basic bit operations

1.3.1 Sum of 3 bits - 3-bit-sum

Given n_1, n_2 their sum produces $n_1 + n_2$ and their carry.
Since $n_1, n_2 \in [0, 1]$, then this operation can be done in $O(1)$.

1.3.2 Summation of 2 numbers

Given n_1, n_2 their sum produces $n_1 + n_2$.
Since the sum is computed bit by bit, the 3-bit-sum is performed $max\{lenght(n_1), length(n_2)\}$ times.
Each time the carry on of the previous sum is added to the two digits.
This operation has then complexity $O(max\{lenght(n_1), length(n_2)\}) = O(max\{log(n_1), log(n_2)\})$

1.3.3 Summation of n numbers

The summation of n numbers is simply the sum of two numbers, but performed $n-1$ times.
Let's assume that $\forall i \in [1, n] : M \geq a_i$.
The complexity of this operation is then $O((n-1) \cdot log(M)) = O(n)$.

1.3.4 Product of 2 numbers

If we consider the classic implementation of the binary multiplication, that is just a sequence of summations.

- The number of summations to execute is equal to the length of the smallest number, $O(log(n))$.
- The maximum cost of a single summation is $O(log(m))$.
- Then, $T(m \cdot n) = O(log(m) \cdot log(n))$, but, if we consider the worst case[2], that becomes $O(log^2(m))$.

[2] two numbers that are equally large

1.3.5 Division of 2 numbers

Let's consider the division of two numbers m, n. This operations consists in finding two numbers q, r such that $m = q \cdot n + r$.
This is achieved by performing a succession of subtractions, until the ending condition $0 \le r < n$ is reached.

- Let's consider that the number of steps of this algorithm is $O(log(q))$.
- Moreover, $q \le m \therefore \#steps = O(log(m))$.
- It's assumed that the cost of the single subtraction is $O(log(n))$.
- Then, $T(\frac{m}{n}) = O(log(n) \cdot log(m))$.

1.3.6 Production of n numbers

Let's assume that $j \in [1, s+1]$ and $M = max(m_j)$.
The cost of the operation $\prod_{j=1}^{s+1} m_j$ is then $O(s^2 \cdot log^2(M))$. This will now be considered our inductive hypothesis.
Proof by induction, on s:

- (1) Base case: $T(m_1 \cdot m_2) = O(log(m_1) \cdot log(m_2)) = O(k_1 \cdot k_2) \le c \cdot k_M^2$.
- (2) Base case: $T(m_1 \cdot m_2 \cdot m_3) = T(m_1 \cdot m_2) + T((m_1 \cdot m_2) \cdot m_3)$
  $\le c \cdot k_M^2 + c \cdot k_{m_1 \cdot m_2} + k_{m_3}$
  $\le c \cdot k_M^2 + c \cdot k_{M^2} + k_M$
- Inductive step: we assume the inductive hypothesis to be true up to s. Then,
  $T(\prod_{j=1}^{s+1} m_j) = T([\prod_{j=1}^{s} m_j] \cdot m_{s+1})$
  $\le c \cdot \sum_{j=1}^{s} (j \cdot k_M^2)$
  $= c \cdot k_M^2 \cdot \frac{s \cdot (s-1)}{2}$
  $= O(k_M^2 \cdot s^2)$
  $= O(s^2 \cdot log^2(M))$

Applications

- An analogous dimonstration can be used to prove that $T(\prod_{j=1}^{s+1} m_j mod n) = O(s \cdot log^2(M))$
- This proof can be used to show that $T(m!) = O(m \cdot log^2(m))$.

1.4 Optimizations of more complex operations

1.4.1 Powers & Modular Powers

Trivial implementation

Square & Multiply method for scalars

Square & Multiply method for polynomials

1.4.2 Finding the b representation of n (n_b)

1.4.3 Modular inverses

1.5 Reminders of Modular Arithmetic

1.5.1 Little Fermat's Theorem

- Let p be a prime number.
- Then, $a^{p-1} \equiv_p 1$.

1.6 Useful Facts

- The GMP library is a free library for arbitrary precision arithmetic. It implements all the basic arithmetic operations with the maximum efficency possible.