Lecture 13 - Basic Number Theory.

Boaz Barak

March 22, 2010

Divisibility and primes Unless mentioned otherwise throughout this lecture all numbers are non-negative integers. We say that A divides B, denoted $A|B$ if there's a K such that $KA = B$. We say that P is *prime* if for $A > 0$, $A|P$ only for $A = 1$ and $A = P$.

Modulu For every two numbers A and B there is unique K and R such that $0 \leq R \leq B - 1$ and $A = KB + R$. In this case we say that $R = A \pmod{P}$. Clearly $B|A$ iff $A \pmod{B} = 0$. Also note that for all A, B, C

$$A + B \pmod{C} = (A \pmod{C} + B \pmod{C}) \pmod{C}$$

and

$$A \cdot B \pmod{C} = (A \pmod{C} \cdot B \pmod{C}) \pmod{C}$$

If $A \pmod{B} = A' \pmod{B}$ we say that A and A' are *equivalent* modulu B, sometimes denoting this by $A \equiv_B A'$.

We denote by $\mathbb{Z}_B$ the set $\{0, \ldots, B-1\}$. When we add or multiply two elements from $\mathbb{Z}_B$ we use addition/multiplication modulu B.

Notation I try to write numbers in capital letters to emphasize the fact that we will normally think of numbers that are of size roughly 2^n where n is our security parameter (= number of bits in our numbers). But I won't always maintain this convention. Generally, an *efficient* operation on n-bit numbers is one that takes poly(n) time. One can verify that the gradeschool addition, multiplication, and long division algorithms, as well as Euclid's gcd algorithm, are efficient.

Unique factorization.

Theorem 1 (Unique factorization). *For every $N > 0$, there are unique primes $P_1, \ldots, P_k$ such as N is the multiplication of these primes.*

We typically order the primes from small to big, and group together multiplications of the same prime, and so the unique factorization of N is its representation of the form $P_1^{i_1} \cdot P_2^{i_2} \cdots P_\ell^{i_\ell}$.

GCD, coprime The greatest common divisor, denoted gcd, of two numbers M and N is the largest number D such that $D|N$ and $D|M$. There is an efficient algorithm to compute D. It can be verified that D is equak to the product of P^i over all primes P that divide both M and N i times (i.e. $P^i|N$, $P^i|M$ but it's not the case that $P^{i+1}|N$ and $P^{i+1}|M$). We say that M and N are co-prime if $gcd(N, M) = 1$. For example, if P, Q, R are distinct primes, $N = PQ^2R$ and $M = Q^2R$ then $gcd(N, M) = Q^2R$.

Basic property of prime and co-prime numbers. Two easy consequences of the unique factorization theorem:

- If P and Q are co-prime and both $P|N$ and $Q|N$, then $PQ|N$.
- If $P|AB$ then either $P|A$ or $P|B$.

Groups A *group* is a set S and operation $\star$ that satisfies associativity: $(a \star b) \star c = a \star (b \star c)$ and inverse: existence of an element id such that for every $a \in S$ there is $a^{-1} \in S$ satisfying $a \star a^{-1} = id$. We denote $a^k = a \star \cdots \star a$ (k times). The group is *Abelian* (also known as *commutative*) if $a \star b = b \star a$ for all $a, b \in S$. We will often use 1 to denote the element id. All the groups in this course will be finite, and I will often forget to mention this explicitly. The size of a group G, denoted $|G|$, is the number of elements in it.

The main group we'll be interested in is the Abelian group $\mathbb{Z}_N^*$ containing the set of all $M \in \{1..N-1\}$ such that $gcd(N, M) = 1$ with the operation being multiplication modulo N. In your homework you will show that it is indeed an Abelian group.

Obviously, if N is prime then $|Z_N^*| = N-1$. Generally $|Z_N^*|$ is denoted as $\varphi(N)$ (this is known as Euler's Phi or totient function) and it's not hard to verify that if $N = P_1^{i_1} \cdots P_\ell^{i_\ell}$ then

$$\varphi(N) = \prod_{j=1}^{\ell} (P_j - 1) P_j^{i_j - 1} = N \prod_{P|N} (1 - 1/P)$$

One thing to note is that if $N = PQ$ where P, Q have roughly equal size (i.e., $P, Q = \Theta(\sqrt{N})$) then $|\mathbb{Z}_N^*| = (P-1)(Q-1) \geq PQ - P - Q$. In particular if we choose a random $A \in \{1...N-1\}$, the probability that $A \notin \mathbb{Z}_N^*$ is at most $P/N + Q/N = O(1/\sqrt{N}) = O(2^{-n/2})$ where n is the number of bits of N.

Order, generators, subgroups If G is finite and $A \in G$, then the sequence $A, A^2, A^3, \ldots$ must repeat itself at some point, meaning that there is some i such that $A^{i+1} = A$ or equivalently $A^i = 1$. The smallest such i is called the *order* of A.

If G is a group and $A \in G$ then $\langle A \rangle$ denotes the set $\{A^i : i \in \mathbb{Z}\}$. This is a also a group with the same operation, and hence it's known as a *subgroup* of G with size the order of A. The size of every subgroup of G divides $|G|$ (homework), and so in particular the order of every element $A \in G$ divides A. A nice corollary for this fact is that for every prime P and $A \in \mathbb{N}$, $A^{P-1} = 1 \pmod{P}$.

An element $A \in G$ is called a *generator* of G if $\langle A \rangle = G$ or equivalently, the order of A is $|G|$.

How many primes exist. Another nice fact to know about primes is that there are infinitely many of them. (It is not immediately obvious from the unique factorization theorem — initially you might think that perhaps the only primes are $\{2, 3, 5\}$ and all other numbers are of the form $2^i 3^j 5^k$). In fact, we have the following theorem:

Theorem 2 (The Prime Number Theorem (Hadamard, de la Vallée Poussin 1896))**.** *For $N > 1$, let $\pi(N)$ denote the number of primes between 1 and N then*

$$\pi(N) = \frac{N}{\ln N}(1 \pm o(1))$$

The original proofs of the prime number theorem used rather deep mathematical tools, and in fact people have conjectured that this is *inherently* the case. But in 1949 both Erdös and Selberg (independently) found elementary proofs for this theorem. For most computer science applications, the following weaker statement proven by Chebychev suffices:

Theorem 3. $\pi(N) = \Theta(\frac{N}{\log N})$

Proof. Consider the number $\binom{2N}{N} = \frac{2N!}{N!N!}$. By Stirling's formula we know that $\log \binom{2N}{N} = (1 - o(1))2N$ and in particular $N \le \log \binom{2N}{N} \le 2N$. Also, all the prime factors of $\binom{2N}{N}$ are between 0 and $2N$, and each factor P cannot appear more than $k = \lfloor \frac{\log 2N}{\log P} \rfloor$ times. Indeed, for every N, the number of times P appears in the factorization of $N!$ is $\sum_i \lfloor \frac{N}{P^i} \rfloor$, since we get $\lfloor \frac{N}{P} \rfloor$ times a factor P in the factorizations of $\{1, \ldots, N\}$, $\lfloor \frac{N}{P^2} \rfloor$ times a factor of the form P^2, etc... Thus the number of times P appears in the factorization of $\binom{2N}{N} = \frac{(2N)!}{N!N!}$ is equal to $\sum_i \lfloor \frac{2N}{P^i} \rfloor - 2\lfloor \frac{N}{P^i} \rfloor$: a sum of at most k elements (since $P^{k+1} > 2N$) each of which is either 0 or 1.

Thus, $\binom{2N}{N} \le \prod_{\substack{1 \le P \le 2N \\ P \text{ prime}}} P^{\lfloor \frac{\log 2N}{\log P} \rfloor}$. Taking logs we get that

$$N \le \log \binom{2N}{N} \le \sum_{\substack{1 \le P \le 2N \\ P \text{ prime}}} \lfloor \frac{\log 2N}{\log P} \rfloor \log P \le \sum_{\substack{1 \le P \le 2N \\ P \text{ prime}}} \log 2N = \pi(2N) \log 2N \ ,$$

establishing $\pi(N) = \Omega(\frac{N}{\log N})$.

To prove that $\pi(N) = O(\frac{N}{\log N})$, we define the function $\vartheta(N) = \sum_{\substack{1 \le P \le N \\ P \text{ prime}}} \log P$. It suffices to prove that $\vartheta(N) = O(N)$ (exercise!). But since all the primes between $N + 1$ and $2N$ divide $\binom{2N}{N}$ at least once, $\binom{2N}{N} \ge \prod_{\substack{N+1 \le P \le 2N \\ P \text{ prime}}} P$. Taking logs we get

$$2N \ge \log \binom{2N}{N} \ge \sum_{\substack{N+1 \le P \le 2N \\ P \text{ prime}}} \log P = \vartheta(2N) - \vartheta(N) \ ,$$

thus getting a recursive equation $\vartheta(2N) \le \vartheta(N) + 2N$ which solves to $\vartheta(N) = O(N)$. $\square$

This means in particular that if you choose a random n-bit integer, with probability $\Omega(\frac{1}{n})$ it will be prime.

Chinese reminder theorem. Let P and Q be two prime numbers (actually can be also just co-prime) and let $N = PQ$. Consider the following function from $\mathbb{Z}_N$ to $\mathbb{Z}_P \times \mathbb{Z}_Q$: $f(X) = \langle X \pmod P, X \pmod Q \rangle$. We claim the following properties of this function:

1. $f(\cdot)$ preserves addition: $f(X+X') = f(X)+f(X')$. (In the right hand side $f(X)+f(X')$ means that we add the first element of both pairs mod P and the second element mod Q. This follows from the fact that the modulu operation has this property.
2. $f(\cdot)$ preserves multiplication: $f(X \cdot X') = f(X) \cdot f(X')$. Again, this follows from the fact that the modulu operation has this property.

3. $f(\cdot)$ is one-to-one. Indeed, if there exist $X > X'$ with $f(X) = f(X')$ then $f(X - X') = \langle 0, 0 \rangle$. Which means that $P|X - X'$ and $Q|X - X'$ which implies $PQ = N|X - X'$ which can't happen for a number between 1 and $N - 1$.
4. $f(\cdot)$ is onto. This follows from the fact that $|Z_N| = |Z_P| \cdot |Z_Q|$.
5. Note that the above properties also imply that f is an isomorphism from $\mathbb{Z}_N^*$ to $\mathbb{Z}_P^* \times \mathbb{Z}_Q^*$.

Operations we can do efficiently We can do the following operations efficiently (polynomial in the number of bits it takes to describe the inputs)

1. Addition and multiplication modulu some N
2. Testing whether $A \in \mathbb{Z}_N^*$ (this is just gcd).
3. Exponentiation modulu N. We can not compute $X^Y \pmod{N}$ by repeated multiplications since that can take Y operation which is too many. Rather we separate Y to a sum of powers of two (binary notation): $Y = 2^i + 2^j + 2^k$ thus we need to compute $X^{2^i} cdot X^{2^j} \cdot X^{2^k}$. We can compute X^{2^i} in i multiplications by repeated squaring.
4. Taking inverse modulu N. If $gcd(X, N) = 1$ then the extended gcd algorithm gives a Y such that $XY \pmod{N} = 1$. We sometimes denote $Y = X^{-1}$.

Non-trivial efficient operations. We'll show we can do the following two things efficiently:

1. Take a square root modulu a prime. That is, for a prime P and $a \in Z_P$, find b such that $a = b^2 \pmod{P}$ if such a b exists.
2. Primality testing: given a number N decide whether it is a prime or a composite number.

Operations we don't know how to do efficiently :

1. Given $N = PQ$, find P, Q.
2. Given $N = PQ$, and $A \in \mathbb{Z}_N^*$, find out if A has a square root modulo N. In particular, we don't know how to find the square root of an A modulo N for A's that do have such square roots.
3. Discrete log: given $G, G^Y \pmod{P}$, find Y.

We note that there are highly non-trivial algorithms for all these operations. For example factoring an n bit integer can be done in time $2^{n^{1/3}\mathrm{polylog}(n)}$. These algorithms are the reason why number theoretic based cryptosystems generally use much larger keys than symmetric cryptography (e.g., 2048 bits vs 128).

Operations we know how to do modulo $N = PQ$ given P, Q :

1. Compute a square root of A modulo $N = PQ$ using the chinese remaindaring theorem.
2. Compute $|Z_N^*| = (P - 1)(Q - 1)$.
3. Compute e^{th} root of A modulo N, assuming $gcd(e, |Z_N^*|) = 1$— this is equivalent to computing $A^{e^{-1} \pmod{|Z_N^*|}}$.

We don't know how to do any of these, and in fact the first two are provably as hard as factoring N.

Fermat's little theorem We'll use the following theorem of Fermat we mentioned above: for every prime P and number $1 \leq A \leq P-1$. $A^{P-1} = 1 \pmod{P}$.

Facts about square roots. When we work in $\mathbb{Z}_P$, we denote by $-X$ the number such that $X - X = 0 \pmod{P}$. In other words, $-X = P - X$. Note that it's always the case that $X \neq -X$ since otherwise we'd have $2X = P$ which means that P is even. We know that over the reals any number a has either zero square roots (if its negative) or two square roots $+\sqrt{a}$ and $-\sqrt{a}$ if its positive. It turns out a similar thing holds for $\mathbb{Z}_P$: every $a \in \mathbb{Z}_P$ has either no square roots, or two square roots of the form X and $-X$.

To prove this first note that if $X^2 = a \pmod{P}$ then $(-X)^2 = a \pmod{P}$. Thus, if a has any square roots it has at least two of them. Now we'll prove that if X and Y are square roots of the same value then $X = \pm Y$. Indeed, if $X^2 = Y^2 \pmod{P}$ this means that $X^2 - Y^2 = 0 \pmod{P}$ or that $P|(X+Y)(X-Y)$. Since P is prime this means that either $P|X+Y$ (meaning $X = -Y \pmod{P}$) or $P|X-Y$ (meaning $X = Y \pmod{P}$).

Taking square root modulu prime: We're given a prime P and a number A which has a square root X, and we want to find X (or $-X$). We can assume P is odd (if P is the only even prime, namely two, then we can easily solve this problem mod P). $P \pmod 4$ can be either 1 or 3. We start with the case that $P \pmod 4 = 3$. That is, $P = 4T + 3$. In this case we claim that A^{T+1} is a square root of A.

Indeed, write $A = X^2$. Then $(A^{T+1})^2 = X^{4(T+1)} = X^{4T+4} = X^{P-1+2} = X^{P-1}X^2 = 1 \cdot A$.

See `http://www.wisdom.weizmann.ac.il/~oded/PS/RND/l11.ps` for the algorithm in the case $P = 1 \pmod 4$. (We note that in that case we use a probabilistic algorithm).

Square roots modulu composites We note the following property about square roots modulu composites: if an odd number N is a product of (powers of) at least 2 distinct primes, then every number a that has square root mod N, has at least 4 square roots. Indeed, if N is of this form then $N = PQ$ for some co-prime P and Q (i.e., P is the power of the first prime, and Q is the rest).

If $X^2 = A \pmod{N}$ then consider the Chinese-remainder function $f(\cdot)$ and denote $f(X) = (X', X'')$ and $f(A) = (A', A'')$. Then, we get that $(X'^2, X''^2) = (A'^2, A''^2)$ but this holds also for all four possible combinations $\langle \pm X', \pm X'' \rangle$.

Primality testing: Let $SQRT(A, P)$ denote our algorithm that on input A, P outputs either "fail" or a number X such that $X^2 = A \pmod{P}$. We'll use that to test whether N is prime. To test whether N is prime, we first check that N is odd and is not a power of some number. If not, we choose a random number $1 \leq X \leq N-1$, compute $A = X^2 \pmod{N}$ and run $SQRT(A, P)$. If it returns "fail" decide that N is a composite. If it returns some number X' such that $X'^2 = A \pmod{P}$ then if $X' = \pm X$ then decide that N is a prime. Otherwise decide that N is a composite.

Theorem 4. *If N is prime then our algorithm finds this with probability at least* 0.99. *If N is composite then algorithm finds this with probability* 0.1.

(Note that we can amplify the success probability of this algorithm using generic techniques.)

Proof. First for our analysis We first make $SQRT$ into a deterministic algorithm by simply choosing coins for $SQRT$ and hardwiring it into to the algorithm. The case of N prime is pretty easy. Suppose N is a composite which is odd and is not a prime power. For every X, say that X is "good" if $SQRT(X^2)$ is either "fail" or is equal to $X' \neq \pm X$. Since there are at least 4 roots for every A, we get that at least two of them are good (there are at most two bad roots for each A). If we hit a good X then we output the right answer. $\square$

Two exercises:

1. Think of the problem of public key cryptography— how can two parties establish confidential communication over a public channel such as the Internet (that can be monitored by routers, wi-fi scanners etc..) without first exchanging a secret key. Then, read the handout on Merkle's 1974 project proposal.

2. (Extra exercise for the mathematically inclined - will be 20 extra points on Homework 7.) Prove that the following algorithm outputs a random number R in $\{1..N\}$ together with R's factorization:

   (a) Generate a random decreasing sequence $N \geq S_1 \geq \cdots \geq S_\ell = 1$, by choosing S_1 at random in $\{1..N\}$, S_2 at random in $\{1..S_1\}$ and so on until reaching 1.

   (b) Let $(P_1, \ldots, P_{\ell'})$ denote the S_i's in this sequence that are *prime*, and let $R = P_1 \cdots P_{\ell'}$. If $R \leq N$ then with probability R/N output R together with its factorization $(P_1, \ldots, P_{\ell'})$.

   (c) If we did not output a number in Step 2b, go back to Step 2a.

   You need to prove that (a) conditioned on outputting a number R, R will be distributed uniformly in $\{1..N\}$ and (b) that the algorithm runs in time poly$(\log N)$. Both follow by showing that for every $R \in \{1..N\}$, the probability that R is output in one iteration of the algorithm in Step 2b is $|Z_N^*|/N^2 = (1/N)\prod_{P|N}(1-1/P)$. (You can use or prove from Chebychev's theorem above the fact that $|Z_N^*| \geq \Omega(N/\log N)$.) See footnote for hint[1]

[1] **Hint:** Think of the random number S_1 as chosen as follows: we output N with probability $1/N$, otherwise we output $N-1$ with probability $1/(N-1)$, otherwise we output $N-2$ with probability $1/(N-2)$, etc..