List of Theorems done in Lectures

ITC1, Spring 2025

by:

Padmini Mukkamala

Budapest University of Technology and Economics

Last updated: September 26, 2025

Note: For the final exam, you have to know the proofs of theorems in a box.

Contents

1 Lecture 1

Divisibility, Euclid's algorithm, Extended Euclid's algorithm.

Divisibility

Proposition 1.1. *If $a|b$ and $b|c$, then $a|c$.*

Proposition 1.2. *If $d|a$ and $d|b$, then $d|ax+by$ for any integers x and y.*

Primes

Lemma 1.3. *Every integer greater than 1 is a prime or divisible by a prime.*

Theorem 1.4. *Euclid's Theorem: There are infinitely many primes.*

Theorem 1.5. *Prime Number Theorem:* $\lim_{x\to\infty} \frac{\pi(x)}{x/\ln x} = 1$

Greatest Common Divisor

Euclidean Algorithm for GCD and Extended Euclidean Algorithm were discussed.

Theorem 1.6. *Let a, b be integers, both not 0. Then there exist integers x, y that can be found by the Extended Euclidean Algorithm such that, $gcd(a, b) = ax + by$.*

Theorem 1.7. *Let $a_1, a_2, ..., a_n$ be integers, not all 0. Then there exist integers $x_1, x_2, ..., x_n$ such that, $gcd(a_1, a_2, ..., a_n) = \sum_{i=1}^{n} a_i x_i$.*

Theorem 1.8. *Let a, b, c be integers such that $a \neq 0$ and $gcd(a, b) = 1$. Then, if $a|bc$, then $a|c$.*

Corollary 1.9. *Let a, b be integers and p a prime. Then, if $p|ab$, then $p|a$ or $p|b$.*

Corollary 1.10. *Let $a_1, a_2, ..., a_n$ be integers and p a prime. Then, if $p|a_1a_2...a_n$, then $p|a_i$ for some $1 \leq i \leq n$.*

Fundamental Theorem of Arithmetic

Theorem 1.11. *Fundamental Theorem of Arithmetic: Every integer $n > 1$ is a prime or can be uniquely written as a product of primes.*

Applications of Fundamental Theorem for the number of divisors of a number and the value of the greatest common divisor and the least common multiple.

2 Lecture 2

Congruence, Linear congruences.

Congruences and their properties

Definition 2.1. *Given integers a, b and $m > 0$, we say that $a \equiv b \pmod{m}$ if both a and b give the same remainder when divided by m.*

Theorem 2.2. *Let $m > 0$ and a, b be integers. $a \equiv b \pmod{m}$ if and only if $m|(a - b)$.*

Congruence classes and Least non-negative residue were illustrated with examples.

Theorem 2.3. *Congruence is an equivalence relation, that is, it is reflexive, symmetric and transitive.*

Theorem 2.4. *Let a, b, c, d, k, m be integers and $k, m > 0$. Further let $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$. Then the following are true:*
a) $a + c \equiv b + d \pmod{m}$
b) $a - c \equiv b - d \pmod{m}$
c) $ac \equiv bd \pmod{m}$
d) $a^k \equiv b^k \pmod{m}$

Proposition 2.5. *Let $m > 0$ and $ac \equiv bc \pmod{m}$. If $gcd(c, m) = 1$, then $a \equiv b \pmod{m}$.*

Proposition 2.6. *Let $m > 0$ and $ac \equiv bc \pmod{m}$. If $gcd(c, m) = d$, then $a \equiv b \pmod{m/d}$.*

Proposition 2.7. *Let a, b, n be integers with $n > 0$. If $a \equiv b \pmod{n}$, then $gcd(a, n) = gcd(b, n)$.*

3 Lecture 3

Linear congruences.

Linear Congruences

Theorem 3.1. *Let a, m be integers with $m > 0$ and $a \neq 0$. If $gcd(a, m) = 1$, then the congruence $ax \equiv b \pmod{m}$ has a unique solution modulo m.*

Proof. (Note: a proof is included here because it is not in the book this way.) There are two steps to proving the statement, one is the existence of a solution, and the second is to prove its uniqueness modulo m.
Existence: Since $gcd(a, m) = 1$, we can find integers x_0, y_0 such that $ax_0 + my_0 = 1$. Multiplying this on both sides with b we get, $abx_0 + mby_0 = b$. Let x be the least non-negative residue of $bx_0 \pmod{m}$. Since $abx_0 + mby_0 = b$, so $abx_0 + mby_0 \equiv b \pmod{m}$, which gives us that $ax \equiv b \pmod{m}$.
Uniqueness: Let x_1 and x_2 be two solutions for the congruence. Then $ax_1 = b + mk_1$ and $ax_2 = b + mk_2$ for some k_1, k_2. Subtracting these two equations, $a(x_1 - x_2) = m(k_1 - k_2)$. But $gcd(a, m) = 1$, so $m|(x_1 - x_2)$ or in other words, $x_1 \equiv x_2 \pmod{m}$. This shows that modulo m, there is a unique solution. □

Theorem 3.2. *Let a, m be integers with $m > 0$ and $a \neq 0$. If $gcd(a, m) = d$, then the congruence $ax \equiv b \pmod{m}$ can be solved if and only if $d|b$. Further, if $d|b$, then there are exactly d solutions of this linear congruence modulo m.*

For the previous theorem, we note that if x_0 is a solution to $(a/d)x \equiv (b/d) \pmod{(m/d)}$, then the d solutions of $ax \equiv b \pmod{m}$ are of the form $x_0 + (m/d) \cdot k$, where $0 \leq k < d$.

System of two linear congruences

Theorem 3.3. *If m and n are relatively prime, then the system of congruences $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$ has a unique solution modulo mn.*

We will further note that the solution to the system of congruences can be found by first finding u and v so that $mu + nv = 1$, and then we set $x = bmu + anv$.

Theorem 3.4. *If $gcd(m, n) = d$, then the system of congruences $x \equiv a \pmod{m}$ and $x \equiv b \pmod{n}$ has a solution if and only if $d|(a - b)$. Further, there is a unique solution modulo $lcm(m, n)$.*

Chinese Remainder Theorem

Theorem 3.5 (Chinese Remainder Theorem)**.** *If $m_1, m_2, ..., m_r$ are positive integers that are pairwise relatively prime, then the system of congruences $x \equiv a_i \pmod{m_i}, 1 \leq i \leq r$ has a unique solution modulo $m_1 m_2 ... m_r$.*

Again we will discuss the form of the solutions. We first find u_i so that $n_i u_i \equiv 1 \pmod{m_i}$, where $n_i = m/m_i$ where $m = m_1 m_2 ... m_r$. Then we set $x = \sum_i a_i n_i u_i$.

4 Lecture 4

Euler-Fermat Theorem.

Euler's Phi Function

Theorem 4.1. *If $a \equiv b \pmod{m}$ then $gcd(a, m) = 1$ if and only if $gcd(b, m) = 1$.*

We introduced the Euler's Phi function $\varphi(n)$.

Lemma 4.2. *If p is a prime and $\alpha \geq 1$ is a positive integer, then $\varphi(p^\alpha) = p^\alpha - p^{\alpha-1}$.*

Lemma 4.3. *If a and b are co-prime positive integers, then $\varphi(ab) = \varphi(a)\varphi(b)$.*

Theorem 4.4. *If $n > 1$ is a integer such that $n = p_1^{\alpha_1} ... p_k^{\alpha_k}$, then,*

$$\varphi(n) = (p_1^{\alpha_1} - p_1^{\alpha_1 - 1}) ... (p_k^{\alpha_1} - p_k^{\alpha_1 - 1}) = n\Pi_{p|n}(1 - \frac{1}{p})$$

.

Euler-Fermat Theorem

We defined a complete residue system (set where every congruence class has exactly one representative) and reduced residue system (set where every congruence class of co-prime numbers has exactly one representative).

Proposition 4.5. *Assume that $R = \{c_1, c_2, ..., c_k\}$ is a reduced residue system modulo m and $a \in \mathbb{Z}$ is an arbitrary integer with $gcd(a, m) = 1$. Then $R' = \{ac_1, ac_2, ..., ac_k\}$ is also a reduced residue system modulo m.*

Theorem 4.6 (Euler-Fermat Theorem)**.** *If a, m are integers, $m > 0$ and $gcd(a, m) = 1$, then $a^{\varphi(m)} \equiv 1 \pmod{m}$ holds, where $\varphi(m)$ is the Euler's phi function.*

Theorem 4.7 (Fermat's Little Theroem)**.** *If p is a positive prime and a is an arbitrary integer, then* $a^p \equiv a \pmod{p}$.

5 Lecture 5

Polynomial Time Algorithms.

We will not be doing any theorems this class!
Below are some algorithms we did in class today.

Addition, slowest.

```
int a,b;
while (a > 0) {
   b = b+1;
   a = a-1;
}
printf("Result: %d", b);
```

Input size: $n = \log_2 a + \log_2 b$. We also checked that the running time of this was bounded from **below** by $2 \cdot (2^{n/2}) = 2 \cdot (\sqrt{2})^n$ steps, thereby showing that this algorithm is not polynomial in its running time.

Addition, faster.

```
int a,b;
while (a > 0) {
   b = b+ ceil(a/2);
   a = floor(a/2);
}
printf("Result: %d", b);
```

Input size: $n = \log_2 a + \log_2 b$. We also checked that the running time of this was bounded from **above** by $5n$ steps, thereby showing that this algorithm is polynomial in its running time.

Gauss' sum, slowest.

```
int a, i, j=0;
for (i=0;i<a;i++) {
   j = j+i;
}
printf("Result: %d", j);
```

Input size: $n = \log_2 a$. We also checked that the running time of this was bounded from **below** by $a = 2^n$ steps, thereby showing that this algorithm is not polynomial in its running time.

LCM, slowest.

```
int a,b;
int x = a;
while (b does not divide x){
   x = x+a;
}
printf("Result: %d", x);
```

Input size: $n = \log_2 a + \log_2 b$. We also checked that the running time of this was bounded from **below** by $b \geq 2^{n/2} = (\sqrt{2})^n$ steps, thereby showing that this algorithm is not polynomial in its running time.

Prime factorization (a little different from class), slowest (but unfortunately, even the fastest is not much faster).

```
int a;
int i = 2, j = ceil(sqrt(a));
while (i<j){
```

```
    if (i|a) {
        printf("%d", i);
        a = a/i;
    }
    else {
        i = i+1;
    }
}
if (a>1){
    printf("Given number is a prime");
```

Input size: $n = \log_2 a$. When a is a prime, we can see that the loop runs $\sqrt{a}$ times. So, the running time of this is bounded from **below** by $2\sqrt{a}2\sqrt{2^n} = 2(\sqrt{2})^n$ steps, thereby showing that this algorithm is not polynomial in its running time.

Modular Exponentiation, slowest.

```
int a,b,m;
int i,x=1;
for (i=1;i <= b;i++) {
    x = x*a mod m;
}
printf("Result: %d", x);
```

Input size: $n = \log_2 a + \log_2 b + \log_2 m$. We also checked that the running time of this was bounded from **below** by $b \geq 2 \cdot 2^{n/3} = 2 \cdot (\sqrt[3]{2})^n$ steps, thereby showing that this algorithm is not polynomial in its running time.

6 Lecture 6

Modular exponentiation, GCD, Prime generation, Primality testing, RSA.

We have a quick discussion on running time of addition and multiplication to be able to give more precise bounds on running times of algorithms. In particular addition can be done in cn time, while multiplication in cn^2 time if n is the size of input.

Modular Exponentiation

Modular Exponentiation.

```
int a,b,m;
int x=1;
while (b>0) {
    if (b is odd) {
        x = x*a mod m;
    }
    b = floor(b/2);
    a = a*a mod m;
}
printf("Result: %d", x);
```

Input size: $n = \log_2 a + \log_2 b + \log_2 m$. Each loop performs at most two multiplications and three divisions of numbers smaller than m^2, this takes at most $c(2\log_2 m)^2 \leq cn^2$ steps. The loop runs $\log_2 b \leq n$ times, so total running time is bounded cn^3 steps, thereby showing that this algorithm is polynomial in its running time.
Note: We did a more precise running time here. If we did the usual computation with basic operations being a step, then we will get cn as the running time. We discussed that this is misleading because then even without $\pmod m$, the running time will be linear, while the running time in this case, computed more precisely with the size of the numbers is exponential.

Euclidean algorithm, Linear Congruences

GCD

```
int a,m; % where 0<a<m
int r=1;
while (r>0) {
   r = m mod a;
   m = a;
   a = r;
}
printf("Result: %d", m);
```

Input size: $n = \log_2 a + \log_2 m$. Each loop performs one division of numbers smaller than m, so with $c(\log_2 m)^2 \leq cn^2$ steps. There are at most $2\log_2 a \leq 2n$ loops. So total running time is bounded cn^3 steps, thereby showing that this algorithm is polynomial in its running time.

Proposition 6.1. *The Euclidean algorithm for gcd stops after at most* $2\lceil \log_2 a \rceil$ *steps.*

From this we deduce that the Euclidean algorithm works in polynomial time.

Proposition 6.2. *In the Extended Euclidean algorithm that for positive integers* a, b *with* $gcd(a, b) = d$*, returns integers* x, y *such that* $ax + by = d$*, the absolute values of* x *and* y *are bounded as follows:* $|x| \leq b/d$ *and* $|y| \leq a/d$*.*

From the above proposition, it follows that the extended Euclidean algorithm also works in polynomial time. From this it also follows that solving linear congruences also can be done in polynomial time, because we solve the congruences by finding modular inverse, which is nothing but an application of the extended Euclidean algorithm.

Primality Testing

We defined Fermat test to decide if a number m is prime, Fermat witness (a number a such that $gcd(a, m) = 1$ but $a^{m-1} \not\equiv 1 \pmod{m}$), and a Fermat liar (a number a such that $gcd(a, m) = 1$ and $a^{m-1} \equiv 1 \pmod{m}$, but m is composite).

Theorem 6.3. *If* m *is a composite positive integer and has a Fermat witness, then at least half of the numbers between* 1 *and* m *that are co-prime to* m *are Fermat witnesses.*

We defined Carmichael numbers. eg. 561, 1105.
Please note: Miller-Rabin test is not in the syllabus. But interested students are encouraged to read it from the ITC1 lecture notes file.

RSA

We discussed the RSA algorithm.

Lemma 6.4. *For any prime* p *and integers* x *and* $k > 0$, $x^{k(p-1)+1} \equiv x \pmod{p}$.

Proof. Notice that the proof is identical to that of Fermat's little theorem. If $p|x$, then $p|(x^{k(p-1)+1} - x)$, and if $p \nmid x$, then $x^{p-1} \equiv 1 \pmod{m}$ from which the given statement follows. □

Theorem 6.5. *Let* p *and* q *be distinct positive primes and* $N = pq$*, then for all integers* x *and* $k > 0$, $x^{k\varphi(N)+1} \equiv x \pmod{N}$.

Proof. Using the lemma, $p|(x^{k(q-1)(p-1)+1} - x)$, and also $q|(x^{k(q-1)(p-1)+1} - x)$. So, $pq = N|(x^{k(q-1)(p-1)+1} - x)$. And we know that $\varphi(N) = (p-1)(q-1)$. □

From here we follow the setup of RSA exactly from the book on page 208,209.

7 Lecture 7

Vectors, Linear combination, Lines, Planes.

We looked at vectors and addition and scalar multiplication of vectors. Then we looked at length of vectors and dot product. Then we looked at linear combination of vectors. In particular, different linear combinations of a single vector gives a line. Linear combination of two vectors not on a line gives a plane. We looked at the equation of line and plane in two, then three dimensions.

Theorem 7.1.

Proposition 7.2.