MA132: FOUNDATIONS, LECTURE NOTES FOR SECOND HALF OF MODULE

DAVID WOOD

These lecture notes have not been previously used, and will be being typed up and adjusted as the module is being taught, so are very much a work in progress. Please inform me via an email (david.wood@warwick.ac.uk), or via the Moodle forums, of any obvious errors.

Still to do

Contents

18. Factors and Prime Numbers

Notation 18.1. *We donote the set* $\mathbb{N}_*$ *to be* $\mathbb{N}\backslash\{0\}$*, i.e. the natural numbers but not containing* 0*.*

Remember that we say a *divides* b, $a \mid b$ if $b = ac$ where $a, b \in \mathbb{N}_*$ for some $c \in \mathbb{N}_*$. We say a *is a factor of* b, and b *is a multiple of* a.

Definition 18.2. *We say* p *is a* prime number *if it has exactly two factors,* 1 *and* p *(so* 1 *is not a prime number).*

Definition 18.3. *If for* $a, b \in \mathbb{N}$ *we have* $c \mid a$ *and* $c \mid b$ *for some* $c \in \mathbb{N}$ *then* c *is a* common factor *of* a *and* b*. If* c *is the largest such factor of* a *and* b *then it is the* highest common factor*:*

$$\mathrm{hcf}(a, b)$$

(sometimes called the greatest common denominator $gcd(a, b)$*).*

For example, 12 and 18 have common factors 1, 2, 3 and 6, so the highest common factor of 12 and 18 is 6.

Definition 18.4. *If the highest common factor of* a *and* b *is* 1*, then* a *and* b *are said to be* coprime*.*

Definition 18.5. *If for* $a, b \in \mathbb{N}$*, the smallest positive* $m \in \mathbb{N}$ *such that* $a \mid m$ *and* $b \mid m$ *is called the* lowest common multiple *of* a *and* b

$$\mathrm{lcm}(a, b).$$

For example, 12 and 18 has $\mathrm{lcm}(12, 18) = 36$.

Theorem 18.6. *Suppose* $a, b \in \mathbb{N}$ *then we have*

$$\mathrm{hcf}(a, b) \times \mathrm{lcm}(a, b) = a \times b.$$

We do not prove this here, since the proof requires prime factorisation which we have not yet properly done. We may return to this proof later, or try it yourself assuming that every number has a unique prime factorisation.

We recall a result that was on Exercise Sheet 5 for completeness:

Lemma 18.7. *Suppose* $a, b, c \in \mathbb{N}$*. If* $a \mid b$ *and* $a \mid c$ *then* $a \mid (b + c)$

Proof. If $a \mid b$ then there exists a $d \in \mathbb{N}$ such that $b = ad$, if $a \mid c$ then there exists $e \in \mathbb{N}$ such that $c = ae$. Then $b + c = ad + ae = a(d + e)$ so $a \mid (b + c)$. □

18.1. Division with Remainder.

Theorem 18.8. *If* $a \in \mathbb{Z}$ *and* $b \in \mathbb{N}_*$ *then* $\exists!$ $r, q \in \mathbb{Z}$ *such that*

$$a = bq + r$$

where $0 \leq r < b$*. I.e. divide* a *by* b *gives* q *times* a *plus remainder term* r*.*

Proof. Fix $b \in \mathbb{N}_*$, prove existence of q, r by induction on a. $P(a)$ is the statement "there exists q, r such that $a = qb + r$ where $0 \leq r < b$". We can assume $b > 1$ since if $b = 1$ then $a = ab$ so $q = a$ and $r = 0$ so we are done.

$P(1)$ is true since we can choose $q = 0$ and $r = 1 < b$ and $1 = 0b + 1$.

Suppose $P(k)$ is true so that there exists a q and $0 \leq r < b$ such that $k = qb + r$. Consider $k + 1 = qb + r + 1$. Either $r + 1 = b$ or $r + 1 < b$. In the first case take $k + 1 = (q + 1)b + 0$ and in the second $k + 1 = qb + (r + 1)$. So $P(k + 1)$ is true. Hence by induction it is true for all $a \in \mathbb{N}_*$. The choice of b was arbitrary so true for all $a,\ b \in \mathbb{N}_*$. □

You may be thinking that this is obvious, so let's look at a different scenario where this Theorem does not hold. Consider $2\mathbb{Z} = \ldots,\ -4,\ -2,\ 0,\ 2,\ 4,\ \ldots$ (the even numbers). A number in $2\mathbb{Z}$ is prime if it is not a product of two smaller even numbers. The first few positive prime numbers are $2,\ 6,\ 10,\ 14$ (anything not divisible by four). So $6,\ 18,\ 30,\ 90$ are all prime.

$$540 = 6 \times 90 = 18 \times 30$$

so note we do not have unique factorisation. If $a = 6$ and $b = 4$ then $6 = 4 \times s + r$ where $0 \leq r < s$ cannot be found with $r,\ s \in 2\mathbb{Z}$ so we do not have division with remainder.

How do we find the highest common factor $\mathrm{hcf}(a, b)$? If we knew we could factorise $a,\ b$ into primes then we are done. For example, to find $\mathrm{hcf}(84, 98)$ note that $84 = 2^2 \times 3 \times 7$ and $98 = 2 \times 7^2$ so the highest common factor $\mathrm{hcf}(84, 98) = 2 \times 7 = 14$. But we will be using highest common factor to prove prime factorisation later, so instead try repeated division with remainder.

Example 18.9. *Apply repeated division with remainder to 81 and 51*

$$\begin{aligned} 81 &= (51 \times 1) + 30 \\ 51 &= (30 \times 1) + 21 \\ 30 &= (21 \times 1) + 9 \\ 21 &= (9 \times 2) + 3 \\ 9 &= (3 \times 3) + 0 \end{aligned}$$

Note that we can also work the above backwards

$$\begin{aligned} 3 &= 21 - (9 \times 2) \\ 3 &= 21 - 2 \times (30 - (21 \times 1)) \\ 3 &= (3 \times 21) - (2 \times 30) \\ 3 &= 3 \times (51 - (1 \times 30)) - (2 \times 30) \\ 3 &= (3 \times 51) - (5 \times 30) \\ 3 &= (3 \times 51) - 5 \times (81 - (1 \times 51) \\ 3 &= (8 \times 51) - (5 \times 81) \end{aligned}$$

We claim that $\mathrm{hcf}(81, 51) = 3$.

Definition 18.10. *Given numbers* $a,\ b \in \mathbb{Z}$ *an* integer linear combination *of* a *and* b *is something of the form*

$$xa + yb$$

for some $x,\ y \in \mathbb{Z}$

Theorem 18.11. *If* $h = \mathrm{hcf}(a, b)$ *and* $d = xa + yb$ *for some* $x, y \in \mathbb{Z}$ *then* $h \mid d$.

Proof. If $h = \text{hcf}(a, b)$ then $h \mid a$ and $h \mid b$. Therefore $a = hm$ for some $m \in \mathbb{Z}$ and $b = hn$ for some $n \in \mathbb{Z}$. Therefore

$$d = ax + by = mhx + nhy = h(mx + ny)$$

so $h \mid d$. □

In general, if $\text{hcf}(a, b) = 1$ then given any $n \in \mathbb{N}$we can find an x and y in $\mathbb{Z}$ so that $n = xa + yb$.

Definition 18.12. *For $a,\ b \in \mathbb{N}$, if $\text{hcf}(a, b) = 1$ then a and b are called* co-prime.

19. Euclids Algorithm

19.1. **Algorithms.** An algorithm consists of

- An input(s), assumed to be a string of length n,
- a set of rules that are repeated, taking the input and providing an output that becomes the next input,
- an output such that the algorithm terminates in finite time (usually a number or another string).

Can we generalise the process from 18.9 to find the highest common factor of two numbers?

19.2. **Euclid's Algorithm.** Given $a,\ b \in \mathbb{N}$ with $a > b > 0$ (re-ordering if necessary) we can write

$$\begin{aligned} a = &\quad q_1 b + r_1 \quad && \text{where } b > r_1 \geq 0 \\ b = &\quad q_2 r_1 + r_2 \quad && \text{where } r_1 > r_2 \geq 0 \\ r_1 = &\quad q_3 r_2 + r_3 \quad && \text{where } r_2 > r_1 \geq 0 \end{aligned}$$

in general $r_t = q_{t+2} r_{t+1} + r_{t+2}$ with $r_{t+1} > r_{t+2} \geq 0$.

Theorem 19.1 (Euclid's Algorithm)**.** *Given the above algorithm on $a,\ b$:*

(1) It stops. That is $\exists k \in \mathbb{N}$ such that $r_{k-1} = q_{k+1} r_k + r_{k+1}$ and $r_k = q_{k+2} r_{k+1} + 0$,
(2) In this case $r_{k+1} = \text{hcf}(a, b)$,
(3) $\exists x, y \in \mathbb{Z}$ such that $\text{hcf}(a, b) = xa + yb$.

Proof. We consider each case in turn.

(1) Note that we have $b > r_1 > r_2 > r_3 > \ldots \geq 0$, the r_i are getting "strictly smaller". Each $r_i \in \mathbb{N}$, $r_i \to 0$ as i increases, so eventually there must be a $k \in \mathbb{N}$ such that $r_{k+2} = 0$.
(2) Consider one step in the algorithm

$$r_i = q_{i+2} r_{i+2} + r_{i+2}$$

we show common factors of $\{r_i, r_{i+1}\}$ are the same as for $\{r_{i+1}, r_{i+2}\}$. This is since
(a) if $m \mid r_i$ and $m \mid r_{i+1}$ then $r_{i+2} = r_i - q_{i+2} r_{i+1}$ so $m \mid r_{i+2}$,
(b) if $n \mid r_{i+1}$ and $n \mid r_{i+2}$ then $r_i = q_{i+2} r_{i+1} + r_{i+2}$ so $n \mid r_i$.
So every step of Euclid's Algorithm preserves the set of common factors, so preserve the highest common factor. The final pair of remainders is $\{r_k, r_{k+1}\}$ but $r_k = q_{k+2} r_{k+1} + 0$ so $\text{hcf}(r_k, r_{k+1}) = r_{k+1}$ and so $\text{hcf}(a, b) = r_{k+1}$.

(3) Finally,we have that

$$r_{k+1} = r_{k-1} - qk + 1r_k$$

so r_{k+1} is a linear combination of r_{k-1} and r_k. Assume there exists $x, y \in \mathbb{Z}$ such that

$$r_{k+1} = xr_{k-t} + yr_{k-t+1}$$

then

$$r_{k-t-1} = q_{k-t+1}r_{k-t} + r_{k-t+1}$$

so

$$r_{k-t+1} = r_{k-t-1} - q_{k-t+1}r_{k-t}$$

giving

$$r_{k+1} = xr_{k-t} + y[r_{k-t-1} - q_{k-1+1}r_{k-t}]$$

which is equal to

$$[x - yq_{k-t+1}]r_{k-t} + yr_{k-t-1}$$

the bit in square brackets is just an integer, so r_{k+1} is a linear combination of r_{k-t} and $rk - t - 1$, so by principle of mathematical induction we can write r_{k+1} as a linear combination of any consecutive pair of remainders. In particular $\text{hcf}(a, b) = r_{k+1}$ is a linear combination of a and b.

□

19.3. **Geometric interpretation of Euclid's Algorithm.** We can think of Euclid's Algorithm geometrically. Say we want to find $\text{hcf}(a, b)$ then consider a rectangle of length a and height b. Remove the largest possible square from this rectangle, which will be a $b \times b$ square, then with the rectangle that remains repeat. Keep repeating until you only have a square left.

Example 19.2. *Consider* $\text{hcf}(24, 15)$ *EVENTUALLY ADD DIAGRAM HERE!*

The final part of the Euclid's Algorithm Theorem is known as Bezout's Lemma:

Theorem 19.3 (Bezout's Lemma)**.** *Let* a, b *be positive integers. Then* $\text{hcf}(a, b)$ *is the smallest positive integer so that it can be written in the form* $xa + yb$ *where* $x, y \in \mathbb{Z}$*.*

Corollary 19.4. *If* a *and* b*, natural numbers, are coprime, then any positive integer can be written as* $xa + yb$ *for some* $x, y \in \mathbb{Z}$*.*

Earlier we saw that every natural number can be divided by a prime number. We now show, given the above, a similar result that we will find useful

Theorem 19.5. *If* p *is a prime number,* $a, b \in \mathbb{N}$ *with* $p \mid ab$*, then either* $p \mid a$ *or* $p \mid b$ *(or both).*

Proof. We will show that if $p \mid ab$ and $p \nmid b$ then $p \mid a$. If $p \nmid b$ then $\text{hcf}(p, b) = 1$ so $\exists x, y \in \mathbb{Z}$ such that $1 = xp + yb$ therefore

$$a = xap + yab$$

but $p \mid (xap)$ since p is an explicit factor, and $p \mid (yab)$ since by hypothesis $p \mid ab$, so therefore $p \mid (xap + yab)$ and so $p \mid a$ proving the result.

□

20. Fundamental Theorem of Arithmetic

We are now in a position to state and prove a theorem we have been alluding to for a while (which is contained in Part 1 of the notes, but we have left to here).

Theorem 20.1 (Fundamental Theorem of Arithmetic)**.** *Every $n \in \mathbb{N}$ can be written uniquely as a product of primes.*

Proof. We prove by induction

(1) (Can be factorised) by (strong) induction $P(n)$ is the statement "$\forall 2 \leq m \leq n$, m can be factorised into primes".

$P(2)$ is true since $2 = 2$. Suppose $P(k)$ true, all $2 \leq m \leq k$ can be factorised into primes, then

- either $k+1$ is prime, whence $k+1 = k+1$ is the factorisation into primes
- or $(k+1) = ab$ where $a, b \in \mathbb{N}$ and $2 \leq a, b \leq k$. So by inductive hypothesis each a, b is a product of primes, so $(k+1) = ab$ gives $k+1$ as a product of primes.

(2) (Can be factorised uniquely) Suppose that

$$n = p_1 p_2 \dots p_r = q_1 q_2 \dots q_m$$

where all the p_i and q_i are primes. First consider p_1 divides $q_1 q_2 \dots q_m = n$ (p_1 divides n). Either $p_1 \mid q_1$ or $p_1 \mid q_2 q_3 \dots q_m$ ($p \mid ab$ then $p \mid a$ or $p \mid b$). In the first case $p_1 = q_1$ since q_1 is prime. In second case $p_1 \mid q_2$ or $p_1 \mid q_3 q_4 \dots q_m$. Continuing we conclude that $p_1 = q_i$ for some i. Without loss of generality say $p_1 = q_1$, reordering if necessary. So now

$$p_2 p_3 \dots p_r = q_2 q_3 \dots q_m$$

by dividing by $p_1 = q_1$. Now repeat the argument with p_2. Inductively we get $p_1 = q_1$, $p_2 = q_2$ up to $p_r = q_m$ with $r = m$.

□

Remark 20.2. *Summary of steps we have taken to prove this fundamental result:*

- *Division with remainder $a \in \mathbb{Z}$, $b \in \mathbb{N}, \exists r, s \in \mathbb{Z}$ s.t. $a = bs + r$,*
- *$\mathrm{hcf}(a, b) = c$ using Euclid, then can find $p, q \in \mathbb{Z}$ so that $c = pa + qb$,*
- *Lemma that if p prime, $a, b \in \mathbb{N}$ then $p \mid ab$ implies $p \mid a$ or $p \mid b$,*
- *Prove Fundamental Theorem of Arithmetic.*

Recall earlier Theorem:

Theorem 20.3. *Suppose $a, b \in \mathbb{N}$ then we have*

$$\mathrm{hcf}(a, b) \times \mathrm{lcm}(a, b) = a \times b.$$

We now prove this

Proof. Let $h = \mathrm{hcf}(a, b)$ a so $a = hm$ and $b = hn$ for some $m, n \in \mathbb{N}$ where m and n are coprime (using unique factorisation into primes). So $\mathrm{lcm}(a, b) = mnh$. Now compute $m \times n = hm \times hn = h \times (nmh)$ this is precisely $\mathrm{hcf}(a, b) \times \mathrm{lcm}(a, b)$ as required. □

20.1. **Continued Fractions.** We can look at Euclid's Algorithm in a slightly different way. Suppose $a, b \in \mathbb{N}$ and consider operations on the fraction a/b. Using division with remainder, if $a = bq + r$ then

$$\frac{a}{b} = \frac{bq + r}{b} = q + \frac{r}{b}.$$

Let's take 81 and 51 again as an example.

$$\begin{aligned}
\frac{81}{51} &= 1 + \frac{30}{51} \\
\frac{51}{30} &= 1 + \frac{21}{30} \\
\frac{30}{21} &= 1 + \frac{9}{21} \\
\frac{21}{9} &= 2 + \frac{3}{9} \\
\frac{9}{3} &= 3
\end{aligned}$$

As before we can now work backwards

$$\begin{aligned}
\frac{21}{9} &= 2 + \frac{1}{3} \\
\frac{30}{21} &= 1 + \frac{1}{2 + \frac{1}{3}} \\
\frac{51}{30} &= 1 + \frac{1}{1 + \frac{1}{2 + \frac{1}{3}}} \\
\frac{81}{51} &= 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{2 + \frac{1}{3}}}}
\end{aligned}$$

Such an expression is called a *continued fraction*. If instead of a rational number we have $x \in \mathbb{R}$ we can still carry out the same process, but it will not terminate.

Let $[x]$ denote "integer part of x and x the decimal expansion, then apply the following process:

(1) Start with x
(2) Write down $[x]$
(3) Form $\{x\}$
(4) If $\{x\} = 0$ then stop
(5) Replace x by $1/\{x\}$
(6) Go to step 2 and repeat

Each step will give a better approximation to x. For example, consider $\pi \approx 3.1415026$:

x	$[x]$	$\{x\}$	$1/\{x\}$
3.1415926	3	0.1415926	7.062516
7.062516	7	0.062516	15.995905
15.995905	15	0.995905	1.0041118
1.0041118	1	0.0041118	243.20249
243.20249	243	0.20249	4.9385155

Note that the longer we go on the worse rounding errors get. The above gives the continued fraction

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{243 + \ldots}}}}$$

We write this as $[3; 7, 15, 1, 243, \ldots]$, let's calculate successive stages

$[3;]$	3	3.000000
$[3; 7]$	22/7	3.1428571
$[3; 7, 15]$	333/106	3.1415094
$[3; 7, 15, 1]$	355/113	3.1415929

The simplest possible non-terminating continued fraction is $[1; 1, 1, 1, 1, \ldots]$. Note, by its definition, this is the same as $x = 1 + \frac{1}{x}$ or $x^2 = 1 + x$ so $x = \frac{1+\sqrt{5}}{2}$.

20.2. **Results we will need later.** We can now see a slightly different proof of a result proved in Part 1 of the notes.

Theorem 20.4. *There are infinitely many primes.*

Proof. Suppose not, so there are a finite number of primes $p_1, p_2, \ldots, p_n$ for some n.

Consider $N = p_1 p_2 \ldots p_n + 1$. None of the p_i divide N since this leaves remainder 1, so none of the p_i are in the prime factorisation of N, but the Fundamental Theorem of Arithmetic says we can uniquely factorise N by primes, so there must exist prime(s) in addition to $p_1, p_2, \ldots, p_n$ contradicting the hypothesis (note, we have not said that N is prime, although it could be). □

21. Chinese Remainder Theorem

There are some more results that we are going to need later on.

Theorem 21.1 (Chinese Remainder Theorem)**.** *Suppose $n_1, n_2, \ldots n_k$ are integers, all of which are pairwise coprime (any two are coprime). Then for any integers $a_1, a_2, \ldots, a_k$ there is an integer x satisfying*

$$x \equiv a_i \pmod{n_i} \text{ for } 1 \leq i \leq k.$$

Proof. The proof uses Euclid's Algorithm and Bezout's Lemma.

Using Euclid's Algorithm we find numbers e_i so that $e_i \equiv 1 \pmod{n_i}$ and $e_i \equiv 0 \pmod{n_j}$ for $j \neq i$.

By letting m_i be the product of all the n_j except for n_i then $1 = b_i n_i + c_i m_i$ (since pairwise coprime).

Then $c_i m_i \equiv 1 \pmod{n_i}$ and $c_i m_i \equiv 0 \pmod{n_j}$ for $i \neq j$. Take $e_i = c_i m_i$ and then let

$$x = \sum_{i=1}^{k} a_i e_i.$$

□

Example 21.2. *Find a number x so that $x \equiv 1 \pmod 3$, $x \equiv 4 \pmod 5$ and $x \equiv 2 \pmod 8$.*

We look for e_1, e_2, e_3 so that

$$\begin{aligned} e_1 &\equiv 1 \pmod 3,\ 0 \pmod 5,\ 0 \pmod 8 \\ e_2 &\equiv 0 \pmod 3,\ 1 \pmod 5,\ 0 \pmod 8 \\ e_3 &\equiv 0 \pmod 3,\ 0 \pmod 5,\ 1 \pmod 8 \end{aligned}$$

e_1 needs to be a multiple of $5 \times 8 = 40$, with $e_1 \equiv 1 \pmod 3$ and $e_1 \equiv 0 \pmod{40}$ i.e. for some m, n we need $e_1 = 3m + 1 = 40n$ so $-3m + 40n = 1$. We use Euclid's Algorithm to obtain $40 = 13 \times 3 + 1$ giving $e_1 = 40$.

Now want $e_2 \equiv 1 \pmod 5$ and $e_2 \equiv 0 \pmod{24}$ since $24 = 8 \times 3$. So $e_2 = 5m + 1 = 24n$ or $-5m + 24n = 1$ or $5m' + 24n = 1$. Using Euclid again we find $24 = 4 \times 5 + 4$ and $5 = 4 \times 1 + 1$ so $1 = 5 - (4 \times 1)$ but $4 = 24 - 4 \times 5$ so $1 = 5 - (24 - 4 \times 5) = 5 \times 5 - 24$ so $m' = 5$ or $m = -5$ giving $e_2 = -24$. Similarly we find $e_3 = -15$ (Exercise!).

Thus $x = 1 \times e_1 + 4 \times e_2 + 2 \times e_3 \pmod{120} \equiv -86 \pmod{120} \equiv 34 \pmod{120}$.

We can check this answer by computing $34 \equiv 1 \pmod 3 \equiv 4 \pmod 5 \equiv 2 \pmod 8$.

22. Fermat's Little Theorem and Euler's Theorem

22.1. Multiplication module m.

Definition 22.1. *A number $a \in \mathbb{Z}/m\mathbb{Z}$ is called a* unit *if it has a multiplicative inverse in $\mathbb{Z}/m\mathbb{Z}$.*

Theorem 22.2. *A number $a \in \mathbb{Z}/m\mathbb{Z}$ is a unit iff $\mathrm{hcf}(a, m) = 1$,*

Proof. Straightforward, left as an exercise. □

Note the special cases where m is prime.

Definition 22.3. *The Euler Totient Function, $\phi(n)$, is the number of positive integers less than or equal to n that are relatively prime to n, alternatively, $\phi(n)$ is the number of units in $\mathbb{Z}/n\mathbb{Z}$.*

For example

$$\begin{aligned} \phi(1) &= 1, \\ \phi(12) &= 4\ (1,5,7,11), \\ \phi(p) &= p-1 \text{ for any prime } p. \end{aligned}$$

Theorem 22.4 (Fermat's Little Theorem)**.** *If p is a prime number, a is any integer, then*

$$a^p \equiv a \pmod{p}.$$

Following immediately on from this is the result

Corollary 22.5. *If p is a prime number, a a number relatively prime to p, then*

$$a^{p-1} \equiv 1 \pmod{p}.$$

We prove the corollary.

Proof. Start by listing the first $(p-1)$ multiples of a

$$a, 2a, 3a, \ldots, (p-1)a\ (*).$$

We claim that these are all distinct modulo p. Suppose not, so $ra = sa \pmod{p}$ some r, s. Thus $(r-s)a \equiv 0 \pmod{p}$. Since a is not a multiple of p, it has an inverse $a^{-1} \pmod{p}$ so $(r-s) \equiv 0 \pmod{p}$ i.e. $r \equiv s \pmod{p}$.

But all numbers $1, \ldots, (1-p)$ are all different $\pmod{p}$ so $r = s$ and numbers in $(*)$ are $1, 2, 3, \ldots, (p-1) \pmod{p}$ in some order. Multiply all together

$$(a)(2a)\ldots((p-1)a) \equiv (p-1)!\ a^{p-1}$$

But this is also conjugate to $(p-1)! \pmod{p}$. Dividing by $(p-1)!$ gives the result we require that $a^{p-1} \equiv 1 \pmod{p}$. □

Theorem 22.6 (Euler's Theorem)**.** *If n is any positive integer, a is relatively prime to n, then*

$$a^{\phi(n)} \equiv 1 \pmod{n}.$$

Proof. This is similar to Fermat's Little Theorem. Look at numbers of the form ka where $1 \leq k \leq n$ and $\text{hcf}(k,n) = 1$, all will be different modulo n so are exactly the numbers modulo n relatively prime to n. Multiply them all together

$$\prod_{k=1}^{n}(ka) \equiv a^{\phi(n)} \prod_{k=1,\text{hcf}(k,n)=1}^{n} k \equiv \prod_{k=1,\text{hcf}(k,n)=1}^{n} k \pmod{n}.$$

Divide both sides by $\prod k$ gives $a^{\phi(n)} \equiv 1 \pmod{n}$ as required. □

23. Algorithmic Complexity

23.1. **Big O Notation.** When looking at functions such as $f(x)$ we are often interested in how large the function grows as x grows, and one way of doing this is having another function as an upper bound.

Definition 23.1. *Let $f(x)$ and $g(x)$ be two functions. We say*

$$f(x) = O(g(x))$$

if there is some constant $C > 0$, which does not depend on x, so that

$$|f(x)| \leq Cg(x)$$

or possible this holds for $x > x_0$ for some x_0.

For example $f(x) = x^3 + 7x^2 - 3x + 5 = O(x^3)$.

Roughly speaking, functions that are $O(x^a)$ for some a (polynomial) grow faster than functions that are $O(\log(x))$ (logarithmic), and functions that are $O(a^x)$ (exponential) grow much faster than functions that are $O(x^a)$.

Note that a function $f(x)$ that is $O(x^3)$ will also be $O(x^4)$ etc, we try to find the $g(x)$ that is in some sense the smallest such function that is an upper bound.

Aside: sometimes maybe $f(x)$ is roughly x^2 but is not $O(x^2)$, we may say $f(x) = O(x^{2+\epsilon})$ if for any $r > 2$ we have that $f(x) = O(x^r)$ but maybe not $f(x) = O(x^2)$.

We won't be using the following in this module, but for completeness, and to see why we specifically call the above Big O notation we define

Definition 23.2 (Little o notation)**.** *We say that $f(x) = o(g(x))$ if for any positive constant $C > 0$ there exists an x_0 so that*

$$0 \leq f(x) < cg(x) \ \forall x \geq x_0.$$

This can be thought of by "as $x \to \infty$ we have that $f(x)$ becomes insignificant compared to $g(x)$.

Definition 23.3 (Logarithm)**.** *Let $b > 0$, $b \neq 1$. If $c > 0$ define*

$$\log_b(c)$$

to be the unique real value a such that $b^a = c$.

So $\log(1) = 0$, $\log_b(xy) = \log_b(x) + \log_b(y)$, $\log_b(x^a) = a \log_b(x)$ and $\log_b(1/a) = -\log_b(a)$.

23.2. **Running Times of Algorithms.** We are interested in algorithms designed to solve number theoretical problems, and how their "running time" increases with the size of the input (length of an input string). We count the "steps" that are needed in order to complete a calculation, so in particular a step might be adding or multiplying two single digit numbers. each such step will take a finite time to compute, this adding up to a total running time. We are not interested in exactly how long each step takes, but rather whether or not the algorithm will terminate in a "reasonable time" or not. Let's consider some examples.

23.2.1. *Addition.* Consider adding two 3-digit numbers by an algorithm. Say $a_3a_2a_1 + b_3b_2b_1$ and think how you would do this using pen and paper. We would first add a_1 to b_1 to get d_1 with a carry c_1 (which may be zero). Then we add a_2, b_2 and c_1 to get the next digit of the sum d_2 with another carry c_2 (which may be zero). Finally we add a_3, b_3 and c_2 to get the third digit d_3 and then we have either a three digit answer, or a four digit answer if there is a further carry c_3.

If we assume here that "addition" is a single step, then this algorithm takes 3 steps. It should be clear that if we have two n digit numbers then addition of those numbers would now require n steps. This algorithm is $O(n)$.

Note, if each we say that adding because of the carry we actually have another addition each time then now our example of 3 digit numbers takes 2×3 steps, and so n digit numbers take $2 \times n$ steps. This is still $O(n)$.

Remember that running time takes the worst case sceneario, we could have taking 123 and 045 as our two three digit numbers, where the latter is actually a two digit numbers so the algorithm would terminate in a shorter time.

23.2.2. *Multiplication.* Now conisder multiplying two n-digit numbers, as an example let's assume we have two 2-digit numbers, so we wish to multiply together a_2a_1 and b_2b_1. To do this we multiply in pairs a_1b_1, a_2b1, a_1b_2 and a_2b_2 which is four steps, and then we add, with carry if neccessary the resultant numbers which is another 4.

In general, for two n digit numbers the same process will require n^2 multiplications and $2n$ additions (plus carries). So multiplcaition is $O(n^2 + n) = O(n^2)$.

Note that there are some clever algorithms that can reduce the number of steps by re-using some calculations that have previously been computed, these have running time $O(m)$ where $1 < m < 2$ (m does not need to be integer valued).

23.2.3. *Division.* Division is no worse than multiplication so is also $O(n^2)$ for our purposes. The most common reasoning for this is using Newton-Raphson to get better and better approximations. We want to solve for x in

$$1/x + a = 0.$$

We start with some first guess and then successive iterations given by $x_{n+1} = x_n - f(x_n)/f'(x_n)$ where $f(x) = 1/x - a$. This gives that $x_{n+1} = x_n(2 - ax_n)$ since $f'(x) = -1/x^2$, so we are taking a sequence of multiplications each of which is therefore $O(n^2)$.

23.2.4. *Primality Testing (naive).* We are given a number N and we want to know if it is prime. Note that later on we'll be more interested in finding factors of a given number N which is a related problem, but is more complex since testing if a number is prime necessarily only finds one factor, thus running time of factorisation will be greater than that of prime testing.

The most naive way is to try to divide N by each integer from 2 to $N-1$, for large N this takes as good as N steps, at each step computing a division. We can reduce this since if N is not prime then it has factors a and b in $\mathbb{N}$ with $2 \leq a, b \leq N-1$ so that $N = ab$. At least one of these must be less than or equal to $\sqrt{N}$, so an upper bound on the number of steps required to check primality is $\sqrt{N}$, so algorithm is $O(\sqrt{N})$? Not quite, remember that until now we have been looking at strings of n digits rather than a single number. N has only $n = O(\log N)$ digits so $N = O(10^n)$, so algorithm runs in $O(\sqrt{10^n})$ steps, but each step requires a division which is $O(n^2)$ so the algorithm runs in $O(\sqrt{10^n} \times n^2)$ steps.

So we are already seeing that testing if a number is prime or not, or finding factors of very large numbers, will not be as easy as we would like.

We will return to primality testing later on (including some some easy quick wins to reduce complexity, but will see that we will still have problems factorising in polynomial time).

23.2.5. *Euclid's Algorithm.* Remember that earlier we introduced Fibonacci numbers in the following way: we take the function $f : \mathbb{N} \to \mathbb{N}$ by

- $f(0) = 0$,
- $f(1) = 1$, and
- $f(k+2) = f(k+1) + f(k)$ for all $k \in \mathbb{N}$.

Doing this $f(n)$ is then the n^{th} *Fibonacci number*. We also proved Binet's Formula, that if $\phi = (1+\sqrt{5})/2$ and $\tau = (1-\sqrt{5})/2$ then

$$f(n) = \frac{\phi^n - \tau^n}{\sqrt{5}}$$

and noted that as $n \to \infty$ we have from the above formula that $f(n) \to \phi^n/\sqrt{5}$.

We now discover Fibonacci numbers in an unexpected place, the running time of Euclid's Algorithm. We wish to calculate $\text{hcf}(a, b)$ where $a, b \in \mathbb{N}$ and $a > b \geq 0$.

Lemma 23.4. *If $a > b \geq 0$ and Euclid's Algorithm takes $k \geq 1$ steps to complete, then $a \geq f(k+2)$ and $b \geq f(k+1)$ where $f(k)$ is the k^{th} term of the Fibonacci sequence.*

Proof. We prove by induction on k.

Start with the base case, and Suppose $k = 1$. Then $b \geq 1 = f(2)$ and since $a > b$ we have $a \geq 2 = f(3)$.

We now assume true for $k-1$ steps and that $\text{hcf}(a, b)$ takes k steps. Then

$$\text{hcf}(a, b) = hcf(b, a \pmod b)$$

the latter is the previous step in Euclid's algorithm and takes $k-1$ steps and $b \geq f(k+1)$ and $a \pmod b \geq f(k)$. So we have shown that $b \geq f(k+1)$, so now consider a.

$$b + a \pmod b = b + a - \lfloor a/b \rfloor b) \leq a$$

where $\lfloor a/b \rfloor$ is the floor function (greatest integer less than or equal to a/b), and the final inequality follows since $a > b > 0$ implies that $\lfloor a/b \rfloor \geq 1$. So

$$a \geq b + a \pmod b \geq f(k+1) + f(k) = f(k+2).$$

□

Corollary 23.5. *For any integer $k \geq 1$ if $a > b \geq 0$ and $b < f(k+1)$ then Euclid's Algorithm requires fewer than k steps to calculate* $\text{hcf}(a, b)$.

Consecutive Fibonacci numbers will provide the worst case examples for Euclid's Algorithm,

$$\text{hcf}(f(k+1), f(k)) = hcf(f(k), [f(k+1) \pmod{f(k)}]) = \text{hcf}(f(k), f(k-1))$$

so needs precisely $k-1$ steps. Recall for large k, $f(k) \approx \phi^k/\sqrt{5}$ so number of steps is $O(\log b)$, but as before b will have $O(\log b)$ digits so running time of Euclid's Algorithm is $O(\log(\log b))$ to compare with previous algorithms.

23.2.6. *P=NP (non-examinable).* So far we have been looking at algorithms designed to find solutions to a problem. If such a problem can find a solution (in worst case scenario) in polynomial time, i.e. running time is $O(n^d)$ for some d, input string length n, we say it is P or *class P*. For example, perviously, addition, multiplication, Euclid's Algorithm are all P. Primality testing is not.

Once we have an answer, we may wish to verify that it is true, also by an algorithm. If a solution can be verified in polynomial time then it is called NP (nondeterministic polynomial time). For example, it can be hard to find factors of a number, but once you have found a factor it is relatively easy to show that it is a factor. In particular if we have a 200 digit number that is two 100 digit prime numbers multiplied together, then it is very difficult to find the factors of the composite number, but once you have found them very easy to verify that they do multiply together to get the original number (an $O(n^2)$ problem).

In general, problems that can be solved and verified in polynomial time, can be done so "quickly". If it takes exponential time then it takes a long time (and the longer the input string is the more unlikely it can be done in any reasonable time).

Conjecture 23.6. *(P=NP or P vs NP) Any solution that can be verified in polynomial time can be solved in polynomial time.*

Consensus is that this conjecture is false ($P \neq NP$), but has not been proven. It is one of the Clay Mathematical Institute million dollar problems). If the conjecture is true then factorising numbers can be done in polynomial time (we just haven't found the right algorithm yet).

24. Discrete Logarithm Problem and the Diffee-Hellman Problem

24.1. **Discrete Logarithm.** Suppose $a, n \in \mathbb{Z}$ and we are given a^n. Then

$$n = \log_a a^n$$

We can, in such cases, try to guess n by trial and error, for example take take $a = 3$ and $a^n = 2187$, what is n? Try $3^5 = 243$, this is clearly way too small, so try $3^10 = 59049$ which is way too large, but we now know that $5 < n < 10$. Try $3^6 = 729$, still too small, $3^7 = 2187$ gives the correct answer, so $n = 7$.

But look what happens if we instead multiply modulo some number, we can no longer use trial and error as above to hone in on the correct answer. I.e. if we are told that a^n

(mod m) and want to find n it becomes much harder.

$$\begin{aligned} 3^2 \pmod 5 &= 4 \\ 3^3 \pmod 5 &= 2 \\ 3^4 \pmod 5 &= 1 \\ 3^5 \pmod 5 &= 3 \end{aligned}$$

In general, in group theory speak, if we have a cyclic group G with generator g we call n the discrete logarithm of g^n.

Conjecture 24.1. *There is no polynomial time algorithm for computing discrete logarithms.*

24.1.1. *Modular Multiplication.* We now introduce two problems that are difficult to solve, with an eye on the ultimate aim of this part of the module.

Definition 24.2. *A number a is a* primitive root *modulo n if every number b coprime to a (i.e.* $\mathrm{hcf}(a, b) = 1$*) is congruent to a power of $a \pmod n$.*

I.e. for every $a \in \mathbb{Z}$ coprime to n there is some integer k for which $g^k \equiv a \pmod n$.

As you should have seen in Algebra 1, let the set $(\mathbb{Z}/m\mathbb{Z})^\times$ be the set of the units of $\mathbb{Z}/m\mathbb{Z}$, these are elements that have a multiplicative inverse. This set is a group under multiplication. The primitive roots of $(\mathbb{Z}/m\mathbb{Z})^\times$ are generators for the group.

If p is prime, then $(\mathbb{Z}/p\mathbb{Z})^\times$ has $p-1$ elements and the primitive roots are the generators. For example, consider the multiplication table for $p = 3$:

$\times$	1	2
1	1	2
2	2	1

We have $(\mathbb{Z}/3\mathbb{Z})^\times = \{1, 2\}$ and the primitive root is 2.

$$\begin{aligned} 2^0 &\equiv 1 \pmod 3 \\ 2^1 &\equiv 2 \pmod 3 \\ 2^2 &\equiv 1 \pmod 3 \end{aligned}$$

Similarly, for $p = 5$ we have

$\times$	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

We have $(\mathbb{Z}/5\mathbb{Z})^\times = \{1, 2, 3, 4\}$ with primitive roots of 2 and 3. The element 4 is not since

$$\begin{aligned} 4^0 &\equiv 1 \pmod 5 \\ 4^1 &\equiv 4 \pmod 5 \\ 4^2 &\equiv 1 \pmod 5 \\ 4^3 &\equiv 4 \pmod 5 \end{aligned}$$

If we keep going, the primitive roots for $p = 7$ are 3 and 5, the primitive roots of $p = 11$ are 2, 6, 7 and 8. The number of primitive roots is given by the following result.

Proposition 24.3. *If $(\mathbb{Z}/p\mathbb{Z})^\times$, p prime, has a primitive root, then it has precisely $\phi(p-1)$ of them.*

Proof. If a is a primitive root of $(\mathbb{Z}/p\mathbb{Z})^\times$ then all elements of the group are of the form $a^k \pmod p$ for some $1 \leq k \leq (p-1)$. If $\operatorname{hcf}(p-1,k) \neq 1$ then there will be an $m < (p-1)$ such that $a^k m \equiv a \pmod p$ so cannot generate all elements of $(\mathbb{Z}/p\mathbb{Z})^\times$ and so is not a primitive root. Hence the set $\{a^k | \operatorname{hcf}(p-1,k) = 1\}$ is precisely the set of primitive roots, which has exactly $\phi(p-1)$ elements. □

Problem 24.4 (The Discrete Logarithm Problem - DLP)**.** *Given a cyclic group G, a generator g for G and some element x of G, find n such that $x = g^n$. Or, given $(\mathbb{Z}/p\mathbb{Z})^\times$, where p is prime, $a \in (\mathbb{Z}/p\mathbb{Z})^\times$ a primitive root, and some element $x \in (\mathbb{Z}/p\mathbb{Z})^\times$, find a number n such that*

$$x \equiv a^n \pmod p.$$

Problem 24.5 (The Diffie-Hellman Problem - DHP)**.** *Given a cyclic group G, a generator g for G and two elements g^a and g^b, compute g^{ab}. Or, given $(\mathbb{Z}/p\mathbb{Z})^\times$, with p prime, a a primitive root, $a^x \pmod p$ and $a^y \pmod p$ for integers x and y, compute*

$$a^{xy} \pmod p.$$

If we can solve (DLP) then we can easily solve (DHP) but

Conjecture 24.6. *There is no polynomial time algorithm for solving solving the Discrete Logarithm Problem.*

Example 24.7. *Take $p = 7$ and primitive root $g = 5$. DHP says given $g^a \equiv 4 \pmod 7$ and $g^b \equiv 2 \pmod 7$ what is g^{ab}? In this case we can use brute force to discover that $5^2 \equiv 4 \pmod 7$ and $5^4 \equiv 2 \pmod 7$ so $g^{ab} = 5^8 = 390625 \equiv 4 \pmod 7$, but now think about doing this for a (much) larger prime p and primitive root g. What condition on g would be beneficial?*

Note, choosing $g^a \equiv 1 \pmod 7$ would have been a bad choice because of Euler's Theorem.

25. Cryptography Primer and the Corrupt Postal Service

A good introduction can be found in [2] or [5], we cover the basics to motivate later results.

25.0.1. *Substitution Ciphers.* Other than having a "dictionary" where you simply substitute words for other words (codes), the next simplest way to encode a message is by so called substitution ciphers where we substitute each letter for another letter in some way.

Definition 25.1. *We call the original, readable message, the* clear text*. The same message that has been encoded in some way is called the* cipher text*.*

The well-known Caesar's Cipher (supposedly used by Julius Caesar) does this substitution by a simple shift of letters in the alphabet, for example:

Clear	A	B	C	D	E	...	Z
Cipher	E	F	G	H	I	...	D

If we number each letter in the natural way

A	B	C	D	E	...	Z
0	1	2	3	4	...	25

Then a Caesar Cipher is just replacing the letter corresponding to the number n with the letter corresponding to the number $m = n + k \pmod{26}$ for some k, called the *key*. The recipient of the message then simply does the reverse substitution $n = m - k \pmod{26}$. The above example has $k = 4$. It should be clear that such a cipher is going to be very easy to crack, with only 25 possible non-trivial ciphers.

A better cipher would be to have more random substitutions, for example

Clear	A	B	C	D	E	...	Z
Cipher	G	O	X	D	S	...	A

There are now $26! - 1$ non-trivial ciphers, 25 of which will be Caesar Ciphers of course, and of the others some will be better than others. But we still have a chance to crack such ciphers, especially as the messages get longer, using letter and word frequency. For example, for the English language the frequency of the most common letters are E (12.7%), T (9.1%), A (8.2%), O (7,5%), I (7.0%), N (6.7%), S (6.3%), H (6.1%) and so on. For three letter words one would first guess that it may be "the" or "and" and so on. Similarly we can look for double letters which could be LL, SS etc.

More sophisticated ciphers will not always substitute the same letter for each occurrence of a clear letter. For example, let's say we have a keyword "DAVE" we can do the following:

Clear	M	A	T	H	S		I	S		N	O	T		A		S	P	E	C	T	A	T	O	R		S	P	O	R	T
X	12	0	19	7	18	/	8	18	/	13	14	19	/	0	/	18	15	4	2	19	0	19	14	17	/	18	15	14	17	19
Y	3	0	21	4	3	/	0	21	/	4	3	0	/	21	/	4	3	0	21	4	3	0	21	4	/	3	0	21	4	3
Key	D	A	V	E	D		A	V		E	D	A		V		E	D	A	V	E	D	A	V	E		D	A	V	E	D
$X+Y \pmod{26}$	15	0	14	11	21	/	8	13	/	17	17	19	/	21	/	22	18	4	23	23	3	19	9	21	/	21	15	9	21	22
Cipher																														

Of course, this also improves with longer keywords. We may also want to remove the spaces and split the message into blocks of five to make it even harder to try and guess words:

MATHS ISNOT ASPEC TATOR SPORT.

25.0.2. *Transposition Ciphers.* Transposition ciphers on the other hand will keep the same letters but put them in a different order, in a pre-arranged way. Now the "key" will be how this is done. The easiest way is to take every other letter, or every third. Where there are then gaps we fill with garbage letters.

E.g.

M	T	S	S	O	A	P	C	A	O	S	O	T
A	H	I	N	T	S	E	T	T	R	P	R	X

or

M	H	S	T	P	T	O	P	T
A	S	N	A	E	A	R	O	C
T	I	O	S	C	T	S	R	F

Of course, to make your message even more secure, you can do a combination of substitution and transposition ciphers.

25.0.3. *The Corrupt Postal Service.* Now let's think about sending encrypted messages, when there is the possibility that someone can intercept anything that is sent between two parties. By convention we suppose that Alice wants to send a secret message to Bob, but a third party, Eve can intercept anything sent between the two. We use the example of sending a parcel by post to abstract the idea.

So we suppose that Alice wants to send a parcel to Bob, so that Eve cannot open the box and see what is inside (Bob needs to receive the box as it was sent, Eve does not want Bob or Alice to know that they have seen the contents. The simplest is the below:

```
Alice and Bob both possess the key to a padlock. Alice locks
the parcel with the padlock, on receiving the package Bob can
unlock it. Eve does not have a key so cannot open it.
```

This is the equivalent of the substitution and transposition ciphers above, both Alice and Bob know the key that has encoded the message, so can encrypt and decrypt. Eve does not know the key so cannot. The problem with this, is that it only works if Bob already has the key. If Alice and Bob have never met, then the only way to get the key to Bob is to send it by post which Eve can intercept and make a copy of. We therefore need a way for Alice to send a parcel without having to send the key first.

```
Alice and Bob each possess their own padlock, with their own key.
Alice padlocks the package and sends it to Bob. Bob cannot unlock
Alice's padlock since he doesn't have a key, so also locks the
package with his padlock and sends the package back to Alice.
Alice now removes her padlock, the package is still locked by Bob's
padlock so she sends the package back to Bob again. Bob can now
unlock his padlock and open the package. At no stage can Eve
intercept and open the package.
```

The problem we have with using a similar idea for encryption is that if Alice's key is a function, or operation, f and Bob's key is a function, or operation, g , then we require f and g to commute, which in general it will not. So instead we need a way to transfer the key, so that it can still be intercepted, but even if it is, cannot be used to decrypt the message.

26. Diffie-Hellman Key Exchange

A message is converted into a number (in fact several numbers where each is then encrypted and sent). For example split the message into sequences of 5 characters, convert each character into its ASCII code (two digit numbers in hexadecimal, or three digit numbers in decimal). This gives either a 10 digit number (hex) or 15 digit number (dec). Beauty of ASCII code unlike the naive numbering earlier is that we can encode spaces, upper and lower case letters, numbers and characters. We then require a key in order to encrypt and decrypt these. We then encrypt this message by some method using a key, which is a further (large?) number.

To get around the issues of key exchange, we want a way to pass the keys between Alice and Bob in such a way that it doesn't matter if someone intercepts messages they cannot construct the key used to encrypt the message. One such method is Diffie-Hellman Key exchange.

(1) Alice and Bob between them pick some prime number p (this will typically be large, circa hundred digits long), and some primive root of $\mathbb{Z}/p\mathbb{Z}$, call it g. Both of these are public information.
(2) Now Alice chooses some number a and Bob chooses some number b, they both keep these numbers secret top themselves.
(3) Alice computes $g^a \pmod p$ and sends to Bob, Bob computes $g^b \mod p$ and sends to Alice.
(4) Now Alice can compute $(g^b)^a = g^{ab} \pmod p$ and Bob can compute $(g^a)^b = g^{ab} \pmod p$.

Hence both Alice and Bob can compute $g^{ab} \pmod p$, this is the key to encrypt the message. Even if Eve intercepts g^a and g^b she cannot "easily" find a or b and so calculate $g^{ab} \pmod p$, especially if a, b, p are large (would need to compute $g^k \mod p$ for $0 \leq k < p-2$ and check if it is the same as g^a or g^b).

Example 26.1. *Alice and Bob choose* $p = 37$ *and primitive root* 5*, these are both public. Alice now chooses* $a = 8$*, and Bob chooses some* b*. Alice computes* $5^8 = 390625 \equiv 16 \pmod{37}$*. Bob computes* $5^b \equiv 14 \pmod{37}$*. Alice now computes*

$$5^{ab} = 14^8 = 1475789056 \equiv 26 \pmod{37}.$$

The key is thus 26, then use favourite method to use this key to encrypt a message.

27. Primality Testing

Another major method for key exchange also uses large prime numbers, and factorising large numbers into (two) prime factors. So we'll have a quick look at algorithms for finding prime numbers/factorisation that are better than our naive methods previously (but will still not be polynomial time algorithms). Showing a number is prime, and finding the factors are clearly related problems, but not exactly the same. The former is easier to demonstrate at this level, so we will concentrate on that.

27.1. **Eratosthenes Sieve.** Eratosthenes was a Greek philosopher and mathematician around approximately 300 BC, his most famous discovery was calculating the circumferance of the Earth to within a couple of percent of the actual, but he was also interested in prime numbers and the following is thus named after him. We have already seen a naive method for trying to test if a number N is prime, by trying all possible factors up to $\sqrt{N}$. A related task is to find all possible prime numbers up to N. To do this we note that if p is a prime number then any multiple of p is certainly not (so if looking for factors of a number N if p is not a factor then we also know any multiple of p is also not). So list the numbers from 2 to N, and start by circling 2 as a prime number, and crossing off every multiple of 2. Then go to the next number in the list that is not crossed off (in this case 3), this must be a prime number so circle it and cross off every mutliple of 3, and so on. We do this up until $N/2$, the any number left uncrossed will also be a prime.

There are other sieve methods that are more efficient, but we do not discuss those here.

27.2. **More Number Theory and Wilson's Theorem.** How about more sophisticated methods to test if a number is prime, which may not tell us what the factors are, but at least would tell us if a number is composite or not. For example, recall Fermat's Little Theorem, if p is prime, a any integer then $a^p \equiv a \pmod{p}$ or if a is coprime to p, $a^{p-1} \equiv 1 \pmod{p}$. Thus if a number does not satisfy either of these then it cannot be prime and so must be composite.

This is not an "if and only if" though, there are composite numbers for which these conditions can be satisfied. For example take 341, and a number $a = 2$ then

$$2^{340} \equiv 1 \pmod{341}$$

and 341 is not prime ($341 = 11 \times 31$), but is known as a pseudoprime. There are also so called Carmichael Numbers, which are numbers n such for all a coprime to n we have $a^n \equiv a \pmod{n}$. The smallest such number is 561, but it has been shown that there are an infinite number of these. We will return to a better method to primality test essentially using Fermat's Little Theorem later, but first we need some more theory.

Lemma 27.1. *An integer n is prime iff the only solutions to the equation*

$$x^2 \equiv 1 \pmod{n}$$

are $x \equiv \pm 1 \pmod{n}$.

Proof. Let p be prime. If $x^2 \equiv 1 \pmod{p}$ then this means that $(x^2 - 1) \equiv 0 \pmod{p}$ and so p divides $x^2 - 1$. Using difference of two squares we further have $(x-1)(x+1) \equiv 0 \pmod{p}$. Thus p divides $(x-1)$ or p divides $(x+1)$ so $x = 1$ or $x = -1$ modulo p and no others.

If a number n is composite then there are more solutions by the Chinese Remainder Theorem (we neglect the remainder of the proof for time but show an example next). □

Example 27.2. *We show an example of a composite number that has more than two solutions to $x^2 - 1 \equiv 0 \pmod{n}$ by choosing $n = 77$ where $77 = 7 \times 11$. $(x^2 - 1)$ has roots ± 1*

$(\mathrm{mod}\ 7)$ *and* $\pm 1\ (\mathrm{mod}\ 11)$ *so we have*

$$\begin{aligned}
x \equiv 1 \pmod 7,\ x \equiv 1 \pmod{11} &\implies x \equiv 1 \pmod{77} \\
x \equiv 1 \pmod 7,\ x \equiv -1 \pmod{11} &\implies x \equiv 43 \pmod{77} \\
x \equiv -1 \pmod 7,\ x \equiv 1 \pmod{11} &\implies x \equiv 34 \pmod{77} \\
x \equiv -1 \pmod 7,\ x \equiv -1 \pmod{11} &\implies x \equiv -1 \pmod{77} \equiv 76 \pmod{77}
\end{aligned}$$

So now return to primality testing. If p is prime, then as above, the only square roots of $1\ (\mathrm{mod}\ p)$ are ± 1, so for any $a \in (\mathbb{Z}/p\mathbb{Z})^\times)$ we have that $a \neq a^{-1}$ unless $a = \pm 1\ (\mathrm{mod}\ p)$ (i.e. there are only two numbers that are self-inverses). So in the list $2, 3, \ldots, p-2$ we have each element and its inverse once and once only. Multiply all these together means that they "cancel each other out" in pairs. If p is composite then this canceling does not happen for all numbers, we get something that is divisible by all proper factors of p. Thus we have proved

Theorem 27.3. *(Wilson's Theorem) For an integer $p > 1$ we have*

$$(p-1)! \equiv -1 \pmod p$$

if and only if p is prime.

27.3. **Miller-Rabin Primality Test.** We now return to Fermat's Little Theorem, and using the above result concerning roots of $x^2 - 1$, to describe a probabilistic test for whether a number n is prime or not. This means that we run a sequence of steps that either tells us n is composite, or suggest that n is a prime number with some probability. If we then repeat this exercise it either tells us n is composite or could be prime with increasing confidence.

We suppose that n is a candidate number to be prime, $n > 2$, and choose a number a such that $2 \leq a \leq (n-1)$. Our first step is then to calculate

$$a^{n-1} \pmod n.$$

If this is not congruent to 1 then we are done, n must be composite. Now note that if $a^{n-1} \equiv 1\ (\mathrm{mod}\ n)$ and n is prime, then the square root of a^{n-1} must be congruent to 1 or -1, if not then n is composite. This square root is simply $a(n-1)/2$, since we must have n odd otherwise it is clearly composite. We have two possibilities, $a(n-1)/2 \equiv \pm 1\ (\mathrm{mod}\ n)$ or $a(n-1)/2 \not\equiv \pm 1\ (\mathrm{mod}\ n)$. In the latter case n must be composite, in the former if $a(n-1)/2 \equiv -1\ (\mathrm{mod}\ n)$ we stop, if $a(n-1)/2 \equiv 1\ (\mathrm{mod}\ n)$ and $n - 1 \equiv 0\ (\mathrm{mod}\ 4)$ then we repeat.

In general, assume that we can write $(n-1) = 2^s q$ for some s and odd q, and work your way down the list

$$a^{2^s q},\ a^{2^{s-1} q},\ a^{2^{s-2} q},\ \ldots,\ a^q$$

and stop when the result modulo n is no longer 1. If it's -1 stop and choose another a, if it's anything else stop because n must be composite.

Note that it's easier to work backwards, start with a^q and square until you get either the sequence $-1, 1, 1, \ldots$ in which case n remains a candidate for a prime, or we go from something that is not ± 1 to 1 in which case n must be composite.

For any composite number n, the probability that n passes this test for primality is less than 0.25 for each choice of a, so if we repeat with several a we start to get a high confidence that n is prime. Let's see a couple of examples (second from [2]).

Example 27.4. *Test for $n = 73$ using $a = 3$. Note that $73 = 9 \times 2^3$. Then*

$$\begin{aligned} 3^9 &\equiv 46 \pmod{73} \\ 3^18 &\equiv 72 \pmod{73} \\ 3^36 &\equiv 1 \pmod{73} \end{aligned}$$

But note that $72 \equiv -1 \pmod{73}$ so 73 (unsurprisingly) passes the test with $a = 3$.

Example 27.5. *Now let's test for $n = 57$ using $a = 20$. Note that $56 = 7 \times 2^3$. Then*

$$\begin{aligned} 20^56 &\equiv 1 \pmod{57} \\ 20^28 &\equiv 1 \pmod{57} \\ 20^14 &\equiv 1 \pmod{57} \\ 20^7 &\equiv 20 \pmod{57} \end{aligned}$$

So 57 is composite.

28. RSA

RSA (Rivest-Shamir-Adleman) forms the basis for modern cryptography, often to encrypt keys to swap between parties rather than directly being used to encrypt messages. As before we describe the basic idea, when used in anger there are various other bells and whistles added to make it even harder to break with brute force methods such as looking for patterns in common message components.

As with Diffie-Hellmann Key Exchange we have a private and public keys, but the difference now is that only one party needs a key.

(1) Alice chooses two prime numbers p and q (typically each of at least 100 digits);
(2) Alice then computes $n = pq$ and $\phi(n) = (p-1)(q-1)$;
(3) Then Alice chooses $e \in \mathbb{Z}$ with $1 < e < \phi(n)$ and $\text{hcf}(e, \phi(n)) = 1$ (often $e = 2^{16} + 1$ is used);
(4) Alice then computes $d = e^{-1} \pmod{\phi(n)}$.

The public key is then n and e, the private key is d. The main point here is that only Alice knows p and q and so $\phi(n)$, and it is very difficult to recover p and q from n. Bob wants to send a message to Alice, plaintext message m, a number modulo n (m coprime to p, but p and q are very large). He computes the ciphertext $c = m^e \pmod{n}$ and sends it to Alice.

Alice can now decrypt by computing $m = c^d \pmod{n}$, which is $(m^e)^d \pmod{n} = m^{ed} \pmod{n}$. We also know that $m^{\phi(n)} \equiv 1 \pmod{n}$ by Euler's Theorem. So $m^r \pmod{n}$ depends only on $r \pmod{\phi(n)}$. Since $ed \equiv 1 \pmod{\phi(n)}$ we have that

$$ed = 1 + k\phi(n)$$

for some k. So

$$m^{ed} \equiv m^{1+k\phi(n)} \equiv m \times (m^{\phi(n)})^k \equiv m \pmod{n}.$$

Alice needs to find d, the inverse of $e \pmod{\phi(n)}$ but she can do this because she knows $\phi(n)$, nobody else does. Knowing n it is hard to find $\phi(n)$ unless you know p and q when $\phi(n) = (p-1)(q-1)$.

Example 28.1. *(from* [2]*) Choose* $p = 71$, $q = 89$ *so* $n = 6319$ *and* $\phi(n) = 70 \times 88 = 6160$.

(1) Alice chooses e *to be coprime to* $\phi(n)$*, for example* $e = 53$*. Then compute* $d = e^{-1} \pmod{\phi(n)}$*. Using Euclid's Algorithm we obtain* $d = 2557$.

(2) Alice sends public key $n = 6319$ *and* $e = 53$ *to Bob.*

(3) Bob wants to send $m = 2161$*, computes* $c = m^e \pmod{n}$ *which is* $c = 3248$ *and sends* c *to Alice.*

(4) Alice computes $m = c^d \pmod{n}$ *which is 2161.*

References

[1] I. Stewart and D. Tall, *The Foundations of Mathematics*, 2nd Edition, Oxford University Press, 2015.

[2] S. Rubinstein-Salzedo, *Cryptography*, Springer Undergraduate Mathematics Series, 2018.

[3] B. Lynn, *Number Theory*, Stanford lecture notes, `https://crypto.stanford.edu/pbc/notes/numbertheory/`, 1980.

[4] J. Buhler and S. Wagon, *Basic Algorithms in Mumber Theory*, Algorithmic Number Theory, 44, 2008.

[5] C.J. Budd and C. Sanguin, *Mathematics Galore*, Oxford University Press, 2001.