CISC-102
Fall 2016
Week 6

We will see two different, yet similar, proofs that there are infinitely many prime numbers. One proof would surely suffice. However, seeing two different ways of proving the same result is instructive, as it demonstrates that there are often many ways in which to make a mathematical argument. I prefer the first proof, but that's simply a matter of taste. Which proof do you prefer?.

Theorem: There exists a prime greater than n for all positive integers n. (We could also say that there are infinitely many primes.)

Proof: (Given any value n we construct a larger value that is either prime or has a prime factor greater than n.**)**

Consider

$$y = n! \text{ and } x = n! + 1.$$

Let p be a prime divisor of x, such that $p \leq n$. This implies that p is also a divisor of y, because n! is the product of all natural numbers from 1 to n. So we have

$$p \mid x \text{ and } p \mid y.$$

According to one of the divisibility theorems we have

$$p \mid x - y.$$

But x - y = 1 and the only divisor of 1 is -1, or 1, both not prime. So there are no prime divisors of x less than n. And since every integer is either prime or a product if primes, we either have x > n is prime, or there exists a prime p, p > n in the prime factorization of x. □

Theorem: There is no largest prime.

Proof: (Proof by contradiction.)

Suppose there is a largest prime. So we can write down all of the finitely many primes as: $\{p_1, p_2, \ldots, p_\omega\}$.

Now let *n* = $p_1 \times p_2 \times \cdots \times p_\omega + 1$.

Observe that *n* must be larger the p_ω the largest prime. Therefore *n* is composite and is a product of primes. Let p_k denote a prime factor of *n*. Thus we have

$p_k \mid n$

And since pk ∈ $\{p_1, p_2, \ldots, p_\omega\}$ we also have

$p_k \mid (n\text{-}1)$

We know that $p_k \mid n$ and $p_k \mid (n\text{-}1)$ implies that $p_k \mid$ n - (*n*-1) or $p_k \mid 1$. But no integer divides 1 except 1, and 1 is not prime, so $p_k \mid 1$ is impossible, and raises a mathematical contradiction. This implies that our assumption that p_ω is the largest prime is false, and so we conclude that there is no largest prime. □

Greatest Common Divisor

Consider any two integers, a,b, at least one non-zero. If we list the positive divisors in numeric order from smallest to largest, we would get two lists:

a: (1, c1, c2, ... |a|)
b: (1, d1, d2, ... |b|)

Since both lists must contain the number 1, we see that 1 is a common divisor of a and b. Since the greatest divisor of a is |a| and the greatest divisor of b is |b|, we can deduce that amongst the common divisors of a and b, there must be one that is the greatest.

Thus we can say that given two integers a,b, at least one not zero, there is a unique greatest common divisor of a and b.

Computing the greatest common divisor of a non-zero integer a, and 0, is somewhat boring because all non-zero integers divide 0, so the greatest common divisor of a and 0 is always |a|. So let's just assume from now on that neither a nor b is 0.

Example:
Let a = 111, and b = 250. We can construct sorted lists of divisors of a and b yielding:

a: (1, 3, 37, 111)
b: (1, 2, 5, 10, 25, 50, 125, 250)

And by inspection we can deduce that 1 is the greatest common divisor of a and b. When the greatest common divisor of two numbers a,b is 1 we say that a and b are *relatively prime* or *coprime*.

Another example:
Let a = 250, and b = 575. We can construct sorted lists of divisors of a and b yielding:

a: (1, 2, 5, 10, 25, 50, 125, 250)
b:(1, 5, 23, 25, 115, 575)

And by inspection we can deduce that 25 is the greatest common divisor of a and b.

This method of obtaining all divisors of a and b is very computationally intensive, and would make some essential steps of public key encryption schemes un-feasible. Remarkably an algorithm invented by Euclid
(~ 300 BC) finds greatest common divisors in a much more efficient way.

Euclid's Algorithm
Suppose a,b are non-zero integers. We can define a function on the integers:

gcd(a,b)

that returns the greatest common divisor of a and b. It will be convenient to further assume that $|a| \geq |b|$.

Euclid's algorithm to compute gcd(a,b) is way more efficient than computing all the divisors of a and b, and is based on the following observation.

Euclid's Theorem:
Let a,b,q,r be positive integers such that a = qb + r then

gcd (a,b) = gcd(b,r)

For example: a = 575, b = 250.

575 = (2)(250) + 75 (Use long division to get q & r)

So the claim is that gcd(575, 250) = gcd(250,75).

This can be verified by listing the divisors of 250 and 75.

250: (1, 2, 5, 10, 25, 50, 125, 250)
75: (1, 3, 5, 15, 25, 75)

We can now "iterate" this process by renaming a = 250, b = 75 and repeat the previous calculation. That is:

250 = (3)(75) + 25

We can again verify that gcd(250,75) = gcd(75,25).

Let's repeat this again, so a = 75 and b = 25

75 = (3)(25) + 0

so we have gcd(75,25) = gcd(25,0), and we have already seen that the greatest common divisor of any non-zero integer a and 0 is |a|.

Therefore by Euclid's algorithm we have
gcd(575,250) = 25.

NOTE: Euclid's algorithm is given for positive integers. However,

gcd(a,b) = gcd (-a,b) = gcd(a,-b) = gcd(-a,-b)

so there is no loss of generality if we simply focus on positive integers.

Observe that as a side effect of Euclid's algorithm we can always find integers x,y such that gcd(a,b) = ax + by.

This can be illustrated with the previous example.

(1) 575 = (2) 250 + 75 implies 75 = 575 - (2)250
(2) 250 = (3) 75 + 25 implies 25 = 250 - (3)75
(3) 75 = (3) 25 + 0

Now we can write gcd(575,250) = 25 as:

25 = 250 - (3)75 (Using (2) above)
25 = 250 - (3)[575 - (2)250] (Using (1) above)
25 = (7)250 - (3)575 (Simplify)

To prove Euclid's Theorem we will need a preliminary result. (Math convention uses the word "lemma" for preliminary results that are proved in preparation for the proof of the main theorem.

Lemma: If $g \mid a$ and $g \mid b$
then $g \mid (pa + b)$ for all integers p.

Proof: Since $g \mid a$ and $g \mid b$ we can write

(1) $a = p_a g$ and $b = p_b g$.

Replacing the values of a and b in $g \mid (pa + b)$
using equations (1) we get:

$$g \mid (pp_a g + p_b g)$$

which simplifies to:

$$g \mid g(pp_a + p_b)$$

Now it should be clear that g divides $g(pp_a+p_b)$ and thus we conclude that g divides pa + b. □

Theorem: Let a,b,q,r be positive integers such that:
$a = qb + r$, $0 \leq r < b$, then gcd (a,b) = gcd(b,r)

Strategy of the proof: We show that the gcd(a,b) is a common divisor of b & r and that gcd(b,r) is a common divisor of a & b.

Proof:
(0) Let g1= gcd(a,b) and g2 = gcd(b,r).

(1) Observe that g2 | b and g2 | r, so g2 | pb + r for all integers p,
and in particular for q, where a = qb + r.

(a) Therefore, g2 | a, so we have established that g2 is a common divisor of both a and b.

(b) Furthermore, observe that $g2 \leq g1 = gcd(a,b)$

(2) Using the equation a = qb + r we can write r = - qb + a.
g1 | b and g1 | a so use the lemma (with p = -p) to get g1 | -pb + a or g1 | r.

(a) Therefor g1 | r and we have established that g1 is a common divisor of b and r.

(b) Furthermore, observe that $g1 \leq g2 = gcd(b,r)$

(3) $g2 \leq g1$ and $g1 \leq g2$ implies that g1 = g2, so we can conclude that gcd(a,b) = gcd(b,r). □

Euclid's Algorithm in the Python programming language.

```
def euclid_gcd(a,b):

# Assume a >= b > 0

    r = a % b # this returns r such that a = bq + r

    while r > 0:

        a,b = b,r

        r = a % b # this returns r s.t. a = bq + r

    return b
```

NOTE: The % (mod) operator is found in many programming languages and returns the remainder when doing integer division.

We will argue that euclid_gcd(a,b) finds gcd(a,b) assuming that $a \geq b > 0$.

We first argue that the loop terminates, that is r eventually becomes 0. This is easy to see because the remainder when we divide a by b is less than b. The value of r begins positive and always decreases so it eventually must be zero.

The correctness follows from Euclid's theorem.

It can also be shown that this function is extremely efficient when compared to looking at all the divisors of a and b.

Let a = 250, and b = 575. We can construct a prime factorization of a and b.

Prime factorization:
$250 = (2)(5^3)$
$575 = (5^2)(23)$

We can inspect the prime factorization of a and b to obtain a greatest common divisor.

Observe that 5^2 is the greatest number that divides both a and b, that is the gcd(a,b). Using the prime factorizations of a and b is much less efficient than Euclid's algorithm.
Nevertheless, the prime factorization is useful for obtaining other properties of the greatest common divisor.

Least Common Multiple

Given two non-zero[1] integers a,b we can have many values that are positive common multiples of both a & b. By the well ordering principle we know that amongst all of those multiples there is one that is smallest, and this is known as the *least common multiple* of a and b. We can define a function lcm(a,b) that returns this value.

Example: Suppose a = 12, and b = 24,
so we have lcm(a,b) = 24.
In general if a | b then lcm(a,b) = |b|.
At this point it is worth mentioning that if a | b then gcd(a,b) = |a|, and that lcm(a,b) × gcd(a,b) = |ab|.

Example: Suppose a = 13, and b = 24, we have
lcm(a,b) = (13)(24).
In general if a and b are relatively prime, that is, if gcd(a,b) = 1 then lcm(a,b) = |ab|

So when gcd(a,b) = 1, we can observe that
lcm(a,b) × gcd(a,b) = |ab|.

Let a = 250, and b = 575. We can construct a prime factorization of a and b

Prime factorization
$250 = (2)(5^3)$
$575 = (5^2)(23)$

We can inspect the prime factorization of a and b to obtain the least common multiple.

$250 \times 575 = (2)(5^3) \times (5^2)(23) = (5^2) \times (2)(5^3)(23)$

And since gcd(a,b) = 5^2 we can conclude that
lcm(a,b) = (2) (5^3) (23).

So in this case we also have lcm(a,b) × gcd(a,b) = |ab|

[1] Multiples of zero are always zero, so this is a boring case.

Let $p_1, p_2, \ldots, p_k$ denote all of the prime factors of both a and b ordered from smallest to largest. In our example the list of prime factors would be 2,5,23.

Let a_i denote the exponent of prime factor p_i, for i, $1 \le i \le k$, in a prime factorization of a.
In our example $a_1 = 1$, $a_2 = 3$, $a_3 = 0$.
Similarly we define b_i for i $1 \le i \le k$.
In our example $b_1 = 0$, $b_2 = 2$, $b_3 = 1$.
Again referring to our example we have:

$\gcd(a,b) = 2^{\min(1,0)} \times 5^{\min(3,2)} \times 23^{\min(0,1)}$

and,

$\text{lcm}(a,b) = 2^{\max(1,0)} \times 5^{\max(3,2)} \times 23^{\max(0,1)}$.

In general using p_i, a_i, and b_i as defined above we can express this formula as

$$\gcd(a,b) = p_1^{\min(a_1,b_1)} \times p_2^{\min(a_2,b_2)} \times \cdots \times p_k^{\min(a_k,b_k)}$$

and

$$\text{lcm}(a,b) = p_1^{\max(a_1,b_1)} \times p_2^{\max(a_2,b_2)} \times \cdots \times p_k^{\max(a_k,b_k)}$$

One more example

$630 = (2) (3^2) (5) (7)$
$84 = (2^2) (3) (7)$

By inspection we can see that
gcd(630,84) = (2) (3) (7) = 42
And lcm(630,84) $= (2^2) (3^2) (5) (7) = 1260$
Again we have

$$\begin{aligned} 630 \times 84 &= (2) (3^2) (5) (7) \times (2^2) (3) (7) \\ &= (2) (3) (7) \times (2^2) (3^2) (5) (7) \\ &= \gcd(630,84) \times \text{lcm}(630,84) \end{aligned}$$

These ideas lead to the following theorem that is given without proof.

Theorem: Let a,b be non-zero integers, then

$$\gcd(a,b)\text{lcm}(a,b) = |ab|.$$

Factoring vs. GCD

Factoring an integer N into its prime factors will use roughly $\sqrt{N}$ operations.

Computing gcd(N,m) with Euclid's algorithm for
N > m ≥ 0 will use roughly $\log_2 N$ operations.

N	$\log_2 N$	$\sqrt{N}$
1024	10	32
1099511627776	40	1,048,576
1×10^{301}	1000	3.27×10^{150}

The efficiency of Euclid's gcd algorithm is essential for implementing current public key crypto systems that are commonly used for e-commerce applications.

With a "key" decoding an encrypted message using Euclid's algorithm takes about 1000 operations. Without a "key" breaking an encrypted messaged takes about 3.27×10^{150} operations. This amounts to a small fraction of a second for decoding and many millions of years for breaking the encrypted message.

Congruence Relations

We say that a is congruent to b modulo m written as:

$a \equiv b \pmod{m}$

and defined as follows:

$a \equiv b \pmod{m}$ if $m \mid (a-b)$.

For example: $64 \equiv 4 \pmod{2}$ and we can verify that $2 \mid 60$.[2]

Example: Let m = 12. Then we have:

$13 \equiv 1 \pmod{12}$

$17 \equiv 5 \pmod{12}$

Which is familiar to everyone who uses a 24 hour clock.

And we can also have:

$241 \equiv 1 \pmod{12}$

$166 \equiv 10 \pmod{12}$

$120 \equiv 0 \pmod{12}$

Similarly

$90 \equiv 30 \pmod{60}$

$75 \equiv 15 \pmod{60}$

$120 \equiv 0 \pmod{60}$

[2] Observe that 2 | -60 too.

We now show that congruence is an equivalence relation.

Theorem: Let m be a positive integer then

1. For any integer a we have $a \equiv a \pmod{m}$ (reflexive)
2. if $a \equiv b \pmod{m}$ then $b \equiv a \pmod{m}$ (symmetric)
3. if $a \equiv b \pmod{m}$ and $b \equiv c \pmod{m}$
   then $a \equiv c \pmod{m}$ (transitive)

I will prove 3.
Congruence is an equivalence relation.

Proof:
If $a \equiv b \pmod{m}$ then $m \mid (a-b)$, (by definition)
and if $b \equiv c \pmod{m}$ then $m \mid (b-c)$.
And by one of the divisibility theorems we have:

$m \mid (a-b+b-c)$ or, $m \mid (a-c)$ so $a \equiv c \pmod{m}$. □