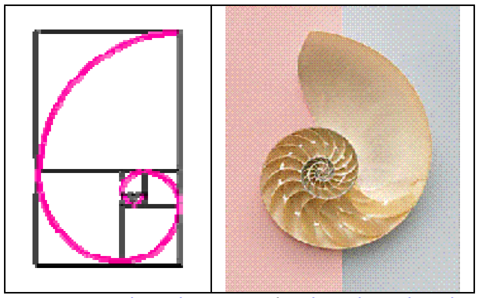
Euler Algorithm – Division Algorithm

Divisors- prime factorization, gcd and lcm

Juan Chamero, jach_spain@yahoo.es lectures
Discrete Structures & Algorithms, ds_008, as of April 2006

..



From Growth and Form and Schenck, Deborah

It's perfectly possible to infer the "core" of this algorithm by inspecting the figure above. Starting with a "seed figure": the first two little squares, you may infer the iteration process more or less like this: "figure shift clockwise 90 degrees pinpointing on first clockwise vertex defining the new figure". Figures are always rectangles. See our Fibonacci Series.

Division Algorithm

Division is not an easy matter as multiplication. In our childhood we were instructed to execute the "Long Division" mechanically without paying to much attention to its inherent complexity.

Given any two integers a, b where b>0, there exists a unique pair of integers [q, r] such that a = b.q + r , where q is called the quotient *of a and b, and r is the remainder.*

Warning: The division algorithm is not an algorithm at all but a theorem. Its name derives from the fact that it was first used empirically as an algorithm to compute the quotient of two integers.

Multiple and Divisor: Given a and b pertaining to the set of integers Z a is called a **multiple** of b if a = b.q for some integer q. We may also say that b is a **divisor** of a using the notation b | a. We may also say that b is a **factor** of a, or that a is **divisible** by b.

Division in Z could be formally stated as: For any integers a and b, with b>0, there exist a unique integer q (the **quotient**) and r (the **remainder**) such that

$$a = b.q + r, \text{ with } 0 \leq r<b.$$

The Euclidean Algorithm

gcd

The **greatest common divisor** or **gcd** of two numbers a and b can be computed by using a procedure known as the **Euclidean algorithm**. First, note that if a $\neq$0 and b | a, then gcd(a, b) = |b|. The next observation provides the basis for the Euclidean algorithm. If a = b.q + r, then (a, b) = (b, r). Thus given integers a>b>0, the Euclidean algorithm uses the division algorithm repeatedly to obtain

$$a = b.q_1 + r_1, \quad \text{with } 0 \leq r_1 < b$$
$$b = r_1.q_2 + r_2, \quad \text{with } 0 \leq r_2 < r_1,$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

Since $r_1 > r_2 > \ldots$, the remainders get smaller and smaller, and after a finite number of steps we obtain a remainder $r_{n+1} = 0$. Thus gcd(a, b) = gcd(b, r_1) = . . . = r_n.

To see how the algorithm works stepwise let's tabulate how Dividend (D), divisor (d), quotient (q) and remainder (r) transform along iterations for these examples: (1525, 125), (137, 9), (1908, 900)

Iter	**D**	**d**	**q**	**r**
1	1525	125	12	**25**
2	25	25	1	0
1	137	9	15	2
2	9	2	4	**1**
3	2	1	2	0
1	1908	900	2	108
2	900	108	8	**36**
3	108	36	3	0

Where in red we depict the corresponding gcd's. Turning back to our childhood let's remember the "factorization" procedure for the (1908, 900) example:

1908	3		900	2
636	3		450	5
212	2		90	5
106	2		18	3
53	53		6	3
1	1		2	2
			1	
1908 = $\mathbf{3^2.2^2}$.53.1			900 = $\mathbf{2^2.3^2}$.5^2	

Where we depict the gcd obtained as the product of two common factors: 4 and 9.

The iteration logic is very simple, namely:

d becomes D
r becomes d

at each iteration and stops when the r=0 condition is attained.

Primes: Nonzero integers a and b are said to be **relatively primes** if (a, b) = 1. Formally this assertion could be stated as: Let a, b be nonzero integers. Then (a, b)=1 if and only if there exist integers m, n such that

$$ma + nb = 1 .$$

With the following properties: for a, b, c integers

(i) If b | ac, then b | (a, b)c.
(ii) If b | ac and (a, b)=1, then b | c.
(iii) If b | a, c | a and (b, c)=1, then bc | a.
(iv) (a, bc)=1 if and only if (a, b)=1 and (a, c)=1.

Prime Numbers: An integer $p>1$ is called a **prime number** if its only divisors are ± 1 and $\pm p$. An integer $a > 1$ is called **composite** if it is not prime, and an integer $p>1$ is prime if and only if it satisfies the following property: If $p \mid ab$ for integers a and b, then either $p \mid a$ or $p \mid b$.

Factorization: Any integer $a>1$ can be factored uniquely as a product of prime numbers, in the form

$$a = p_1^{m_1} p_2^{m_2} \cdots p_n^{m_n}$$

Where $p_1 < p_2 < \cdots < p_n$ and exponents $m_1, m_2, \ldots, m_n$ are all positive. There exist infinite prime numbers.

lcd

The **Least common multiple** or the **lcm** of any two nonzero integers a and b is a number m subject to the following restrictions:

(i) m is a multiple of both a and b, and
(ii) any multiple of both a and b is also a multiple of m.

And we may use the notation lcm [a,b] for the least common multiple of a and b.

Properties: Let a and b be positive integers with prime factorizations

$$a = p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n} \quad \text{and} \quad b = p_1^{b_1} p_2^{b_2} \cdots p_n^{b_n},$$

with $a_i \geq 0$ and $b_i \geq 0$ for all i (allowing use of the same prime factors.)
For each i let $d_i = \min\{a_i, b_i\}$ and let $m_i = \max\{a_i, b_i\}$. Then we have the following factorization scheme:

(i) $\gcd(a,b) = p_1^{d_1} p_2^{d_2} \cdots p_n^{d_n}$
(ii) $\mathrm{lcm}[a,b] = p_1^{m_1} p_2^{m_2} \cdots p_n^{m_n}$

Congruence

Integers a and b are said to be **congruent modulo n,** being n a positive integer if they have the same remainder when divided by n. This is denoted by writing $a \equiv b \pmod{n}$. When working with congruence modulo n, the integer n is called the **modulus**. By the preceding proposition, $a \equiv b \pmod{n}$ if and only if $(a - b) = n.q$ for some integer q. We can write this in the form $a = b + n.q$, for some integer q. This observation gives a very useful method of replacing congruence with an equation (over **Z**). On the other hand any equation can be converted to a congruence modulo n by simply changing the = sign to $\equiv$. In doing so, any term congruent to 0 can simply be omitted. Thus the equation $a = b + n q$ would be converted back to $a \equiv b \pmod{n}$.

Some properties and theorems: Let $n>0$ be an integer. Then the following conditions hold for all integers a, b, c, d:

(i) If $a \equiv c \pmod{n}$ and $b \equiv d \pmod{n}$, then $a \pm b \equiv c \pm d \pmod{n}$, and $ab \equiv c.d \pmod{n}$.
(ii) If $a + c \equiv a + d \pmod{n}$, then $c \equiv d \pmod{n}$
(iii**)** If ac a.d (mod n) and $(a, n)=1$, then $c \equiv d \pmod{n}$.

Let a and n>1 be integers. There exists an integer b such that $a.b \equiv 1 \pmod{n}$ if and only if (a, n)=1.

The congruence $a.x \equiv b \pmod{n}$ has a solution if and only if b is divisible by d, where d = (a, n). If d | b, then there are d distinct solutions modulo n, and these solutions are congruent modulo n / d.

The Chinese Remainder Theorem: Let n and m be positive integers, with (n, m) = 1. Then the system of congruence

$$x \equiv a \pmod{n} \qquad x \equiv b \pmod{m}$$

has a solution. Moreover, any two solutions are congruent modulo m. n.

Congruence Classes: Let a and n>0 be integers. The set of all integers which have the same remainder when divided by n is called the **congruence class** of a modulo n, and is denoted by $[a]_n$, where

$$[a]_n = \{ x \in \mathbf{Z} \mid x \equiv a \pmod{n} \}.$$

All congruence classes modulo n set is called the **module n set of integers**, denoted by $\mathbf{Z}_n$.

Some properties and Theorems: Let n be a positive integer, and let a, b be any integers. Then the addition and multiplication of congruence classes given below are well-defined by:

$$[a]_n + [b]_n = [a+b]_n \,, \qquad [a]_n[b]_n = [ab]_n.$$

If $[a]_n$ belongs to $\mathbf{Z}_n$, and $[a]_n[b]_n = [0]_n$ for some nonzero congruence class $[b]_n$, then $[a]_n$ is called a **divisor of zero**, modulo n.

If $[a]_n$ belongs to $\mathbf{Z}_n$, and $[a]_n[b]_n = [1]_n$, for some congruence class $[b]_n$, then $[b]_n$ is called a **multiplicative inverse** of $[a]_n$ and is denoted by $[a]_n^{-1}$. In this case, we say that $[a]_n$ and $[b]_n$ are **invertible** elements of $\mathbf{Z}_n$, or **units** of $\mathbf{Z}_n$.

Let n be a positive integer. It follows:

(i) The congruence class $[a]_n$ has a multiplicative inverse in $\mathbf{Z}_n$ if and only if (a, n)=1.
(ii) A nonzero element of $\mathbf{Z}_n$ either has a multiplicative inverse or is a divisor of zero.

The following conditions on the modulus n > 0 are equivalent:

(i) The number n is prime.
(ii) $\mathbf{Z}_n$ has no divisors of zero, except $[0]_n$.
(iii) Every nonzero element of $\mathbf{Z}_n$ has a multiplicative inverse.

Totient

Let n be a positive integer. The number of positive integers less than or equal to n which are relatively prime to n will be denoted by $\varphi(n)$. This function is called **Euler's phi-function**, or the **Totient function**.

If the prime factorization of n is $n = p_1^{m_1} p_2^{m_2} \cdots p_n^{m_n}$, then

$$\varphi(n) = n(1-1/p_1)(1-1/p_2) \cdots (1-1/p_k).$$

The set of units of $\mathbf{Z}_n$, the congruence classes $[a]_n$, such that (a, n)=1, will be denoted by $\mathbf{Z}_n^{\times}$.

Another Euler Theorem: If (a, n) = 1, then $a^{\varphi(n)} \equiv 1 \pmod{n}$.

Fermat Corollary: If p is prime, then $a^p \equiv a \pmod{p}$, for any integer a.

Some discussion and properties

In Z and R realms

Suppose p and q are relatively prime integers, and $0 < p < q$. If q has any prime factors besides 2 and 5, then a decimal expansion of p/q will eventually start to repeat. How long will the repeating part be?

Looking at the sequence of digits is actually a very difficult way to approach this problem. It's much simpler to look at the sequence of remainders. Remember how long division works. At every step, we have a remainder r, bounded by 0 <= r < q. To get the next digit in the expansion, we figure out how many times q goes into 10*r; this gives us a digit and a new remainder. In C++ code, we may write the long division algorithm this way:

```
/* Compute n digits of p/q */
r = p;
for (i = 0; i < ndigits; ++i)
            {
            d[i] = (10*r) / q;
            r    = (10*r) % q;
            }
```

The operator % is the mod operator that computes the remainder in a division. If two results are equivalent modulo m, it means that they pertain to the same equivalence class. For example the bit sequence [0 1 2 3] in base 4 represent the infinite sequences [0 1 2 3], [4 5 6 7], [8 9 10 11],……., sequences of numbers that differ by a multiple of the base 4 used as module.

The C code above computes then a sequence of remainders, which look like

$$r[k] = (p*10\string^k) \% q$$

If q has prime factors other than 2 and 5, we are going to find a remainder which is prime relative to q; at that point. From this point onwards the remainders are all units of the ring of integers modulo q (Z_q). For instance, 4 and 7 are relatively prime, so 4 is a unit in Z_7; the inverse is 2, since 4*2 = 7+1 = 1 modulo 7.

Euler Phi Function: The set of units in Z_q is an important object in number theory: the number of natural numbers less than q which are prime relative to q is written phi(q) and is called **Euler's phi function** or **Euler's totient function**.

Division Algorithm script

From Ian Kaplan

```
/*
   Copyright stuff

   Use of this program, for any purpose, is granted the author,
   Ian Kaplan, as long as this copyright notice is included in
   the source code or any source code derived from this program.
   The user assumes all responsibility for using this code.

   Ian Kaplan, October 1996

*/


void unsigned_divide(unsigned int dividend,
                     unsigned int divisor,
                     unsigned int &quotient,
                     unsigned int &remainder )
{
  unsigned int t, num_bits;
  unsigned int q, bit, d;
  int i;

  remainder = 0;
  quotient = 0;

  if (divisor == 0)
    return;

  if (divisor > dividend) {
    remainder = dividend;
    return;
  }

  if (divisor == dividend) {
    quotient = 1;
    return;
  }

  num_bits = 32;

  while (remainder < divisor) {
    bit = (dividend & 0x80000000) >> 31;
    remainder = (remainder << 1) | bit;
    d = dividend;
    dividend = dividend << 1;
    num_bits--;
  }


  /* The loop, above, always goes one iteration too far.
     To avoid inserting an "if" statement inside the loop
     the last iteration is simply reversed. */

  dividend = d;
  remainder = remainder >> 1;
  num_bits++;
for (i = 0; i < num_bits; i++) {
    bit = (dividend & 0x80000000) >> 31;
    remainder = (remainder << 1) | bit;
    t = remainder - divisor;
    q = !((t & 0x80000000) >> 31);
    dividend = dividend << 1;
    quotient = (quotient << 1) | q;
    if (q) {
       remainder = t;
     }
  }
}  /* unsigned_divide */

#define ABS(x)  ((x) < 0 ? -(x) : (x))

void signed_divide(int dividend,
                                             int divisor,
                                             int &quotient,
                                             int &remainder )
{
  unsigned int dend, dor;
  unsigned int q, r;

  dend = ABS(dividend);
  dor  = ABS(divisor);
  unsigned_divide( dend, dor, q, r );

  /* the sign of the remainder is the same as the sign of the dividend
     and the quotient is negated if the signs of the operands are
     opposite */
  quotient = q;
  if (dividend < 0) {
    remainder = -r;
    if (divisor > 0)
      quotient = -q;
  }
  else { /* positive dividend */
    remainder = r;
    if (divisor < 0)
      quotient = -q;
  }

} /* signed_divide */
```