Applied Mathematics for Reversers III

By Haldir[RET]

Two useful algorithms for modular arithmetics

This is the last essay about modular arithmetics, about the Extended Euclidean Algorithm for modular multiplicative inverse calculation and finally about the Chinese Remainder Rheorem (CRT).

1. The Extended Euclidean Algorithm for modular multiplicative inverse calculation

You all know what gcd(m,n) is, the greatest common divisor of m,n, now to the extended version of it.
The egcd(m,n) solves the problem x*m+y*n = gcd(m,n)

Example:

m	n	y	x
24948	8712	-20	7
8712	7524	7	-6
7524	1188	-6	1
1188	396	1	0

So what does this table mean?

The two columns at the left are the usual gcd(m,n) output,you should easily see how the two columns are related,
that the n value of the first row is the m value of the second row and the n value of the second row is m%n from the first row etc.
The last n value in the 4th row is the result of gcd(m,n) (396).

Now to the right two columns:
They solve x*m+y*n = gcd(m,n), you start with the last row and work yourself up to the top:
0*1188+1*396 = 396 , then the y value from the last row is the x value from

the 3rd row
1*7524+(-6)*1188 = 396, y moves up to x again
-6*8712+7*7524 = 396, y moves up to x again
7*24948+(-20)*8712 = 396

This was our example for the egcd(m,n), now where can we use it, well you read the headline, to calculate the multiplicate inverse of a number for modulo.

Example:
You have some piece of code:

```
int check( int b)
{
int a = 8549;
if(a*b==1) return 0;
else return -1;
}
```

You can't do b = 1/a since for integer you'll get most likely b = 0 as a result. So what can we do?

We solve a*b == 1 mod 2^32 (2^32 for integer)
-> b == 1/a mod 2^32 (division is not completely correct, but since we solve it in the residue ring Z/Z*(2^32) we accept it).
This will just work if a and n (in our case 8549 and 2^32) are relatively prime, e.g. gcd(8549,2^32) == 1
There are two cases: n is prime or n is not prime, n is not prime is the harder case (usually ours ;))
for n = prime we can calculate the inverse with a^(p-2) % p == i, e.g. 5^(131-2) % 131 == 105; 105*5 % 131 == 1 :)
Now to the complicated part:
Now to our egcd() algo: x*m+y*n == gcd(m,n)
x*8549+y*2^32 == 1

Since this would be futile to solve manually here is an example with __int8:
x*25+y*2^8 = 1

m	n	y	x
25	256	-4	41
256	25	41	-4
25	6	-4	1
6	1	1	0

(-4*256+41*25) = 1 for the 2nd row and (1*25+(-4)*6) = 1 for the 3rd row

so we know 41*25+(-4)*256= 1; (41*25)%256 == 1
now the inverse of a : a* a^-1 = 1 % n -> 25*41 % 256 -> a^-1 = 41
Result: The inverse of 25 % 256 is 41.

In our 32bit example above the solution is 3509725293;
so 3509725293*8549 % 2^32 == 1 :)

That's it about modular multiplicative inverse calculation.

2. Chinese Remainder Theorem

The Chinese Remainder Theorem is an old Theorem:
For every Integer number m,n with gcd(m,n) = 1 and a is element of Z(m)
and b is element of Z(n), there is a unique
t which is element of Z(mn), that
t == a (mod m) and t == b (mod n);
This is the simple CRT, you can extend it to infinite equations.
I will just discuss the simple version.

For Example You have a piece of code like this:

```
bool check(int number) {
return ((number%127 == 7) && (number % 109 == 11));
}
```

and you want to know the number so the function returns TRUE.

This is a perfect example for the CRT:
number = a[0] % m[0];
number = a[1] % m[1];
number == 7 % 127 and number = 11 % 109

Let's define M, M[i] and N[i]:
M := sum(m[i]) -> M = 127*109 = 13843;
M[i] := M/m[i] -> M[0] = 109 and M[1] = 127;
N[i] := M[i]*N[i] == 1 (mod m[i]) -> N[0] = 7 and N[1] = 103
Here you see why i explained the modular multiplicative inverse above :)

We solve:
number == sum(a[i]*M[i]*N[i]) (mod M); -> number ==
a[0]*M[0]*N[0]+a[1]*M[1]*N[1] (mod M)
-> 7*109*7+11*127*103 (mod 13843) == 10802 :)
Let's see if this is correct: 10802 % 127 == 7 and 10802 % 109 == 11 ->
q.e.d.

There are infinite congruent solutions, if you need more solutions just do rand()*10000*M+10802.
That's it, now you know how to solve the CRT, you can easily extend it to more than two equations.
Now you should know pretty much everything necessary to solve most modular arithmetic problems.

If you want to know more about this topic you should read Victor Shoup's Book about Number Theory,
You can get it here: http://shoup.net/ntb/. It focuses on the computational point of view and contains detailed explanations of all necessary math.

Haldir[RET]
10/25/03

http://www.reteam.org