© The CodeBreakers-Journal, Vol. 1, No. 1 (2004)
http://www.CodeBreakers-Journal.com

Abstract Algebra in Poly Engines

Author: Whitehead

Abstract

In this article I will try to show how to generate an algorithm used in PRIDE method. Simply, we want to generate "random" integer numbers between 0 and n-1 and every one of them should be generated only once. To do this we will use few facts known from algebra and we will achieve very simple formula for PRIDE method. I remark: this in not a simple text to understand.

Keywords: *VX-Knowledge, Polymorphic Engines, Math-Primer, PRIDE*

NOTE:
This article had been published first at the 29A eZine[1] *(Issue #6). Publishing at CodeBreakers-Journal with friendly permission.*

Important note: you should first read article "Advanced polymorphic engine construction" by Mental Driller.

I. Definitions

A. Definition 1

A prime number is an integer p greater than one with the property that 1 and p are the only positive integers that divide p.

B. Definition 2

The greatest common divisor (GCD) of $a_1, a_2, ..., a_r$ is the greatest positive integer that divides each of $a_1, a_2, ..., a_r$. If the greatest common divisor of $a_1, a_2, ..., a_r$ is 1 then these integers are said to be coprime.

[1] http://29a.host.sk/

C. Definition 3

A group G consists of a set G together with a binary operation * for which the following properties are satisfied:

1) $(x * y) * z = x * (y * z)$ for all elements x, y, and z of G (the Associative Law)
2) There exists an element e of G (known as the identity element of G) such that $e * x = x = x * e$, for all elements x of G.
3) For each element x of G there exists an element x' of G (known as the inverse of x) such that $x * x' = e = x' * x$ (where e is the identity element of G).

NOTE: We use char * but it isn't multiply.

D. Definition 4

A group G is said to be cyclic, with generator x, if every element of G is of the form x^n for some integer n.

NOTE: x^n = power n of x.

For example:
We have group with + (add) then $x^3 = x + x + x$)
We have group with * (mul) then $x^3 = x * x * x$)

E. Definition 5

The order $|G|$ of a finite (finite number of elements) group G is the number of elements of G.

F. Definition 6

When I write operation *(mod p), I think about multiply modulo p, and when I write +(mod p), i think... guess what :))

II. Examples

- Integer numbers with operation + (add), (Z,+) is infinite cyclic (number 1 is a generator) group.
- Matrices 2x2 with operation + is infinite group.
- Matrices 2x2 with operation * isn't a group. There aren't inverse elements for matrix: $\begin{bmatrix} 4 & 2 \\ 10 & 5 \end{bmatrix}$
- Integer numbers with operation * (mul), (Z,*) isn't a group, there aren't inverse elements. For example: there is no such b, that 4*b = 1.
- Numbers 1,...,10 with operation *(mod 11) is finite group (cyclic ??? :)

III. Theorem 1 (Fermat)

Let p be a prime number. Then $x^p = x(mod\ p)$ for all integers x. Moreover if x is coprime to p then $x^(p-1) = 1(mod\ p)$.

We have all, let's go to work :)

IV. Simple methods

We will talk about groups 0,...,n-1 with an +(mod n) operation. Every group of this type is cyclic, because it is seen at once, that 1 is a generator of every one of them.

Let n=10. We want to generate numbers 0,1,...,9. Group 0,1,...,9 with an +(mod 10) operation has a generator 1, but it is bad generator. Let's find another one:

2 - is not a generator, because there is no such b, that $2^b \ mod \ 10 = 3$
3 - is a generator

```
1=3^7  mod 10
2=3^4  mod 10
3=3^1  mod 10
4=3^8  mod 10
5=3^5  mod 10
6=3^2  mod 10
7=3^9  mod 10
8=3^6  mod 10
9=3^3  mod 10
0=3^10 mod 10
```

and so on.

(NOTE: Problems ??? Look on NOTE in Definition 4)

It is not clearly seen at once but let's trust our intuition that:

A. Fact 1

G = group (0,...,n-1,+(mod n)), g is a generator if GCD(n,g) = 1

The conclusion is that if n=p is a prime number then every number a ($0 < a < p$) is a generator of the group $0, ..., p-1$ with an $+(mod\ p)$ operation. Let's check the quality of the generator we have found: 3. Let's see which "random" numbers it generates. We have 3, 6, 9, 2, 5, 8, 1, 4, 7, 0. These number are not "random" enough and there is no need to use algebra to generate them.

That's why we finish to talk about the groups with an +(mod p) operation.

V. A litle bit more dificult, but efficient methods.

Now we will concentrate on the multiplicative groups (based on the multiplication operation). What about the sequence:

18, 5, 3, 25, 15, 9, 17, 16, 27, 22, 19, 23, 8, 28, 11, 24, 26, ... ,1 ?

We have here all the numbers greater than 1 and less than 29. I think that it is "random" enough to satisfy the assumptions of PRIDE method. How to generate such a sequence ?

Let's try to generate a "random" sequence from the interval ¡1,9¿ using and *(mod 10).

Let's try to find a generator.

```
* 1 - goes out
* 2 - Hmmm... goes out
* 3 - let's see

     3^1 mod 10 =  3   mod 10 = 3
     3^2 mod 10 =  9   mod 10 = 9
     3^3 mod 10 =  27  mod 10 = 7
     3^4 mod 10 =  81  mod 10 = 1
     3^5 mod 10 =  243 mod 10 = 3  <- the end of the generation
                                      loop, it is seen that 3 isn't
                                      a generator.

       .--> 3 ---> 9 ---> 7 ---> 1 --. We can't generate (2,4,5,6,8)
       '------------------------------'

* 4 - Hmmm...goes out
* 5 - Hmmm...goes out
* 6 - Hmmm...goes out
* 7 - let's see

      7^1 mod 10 =  7     mod 10 = 7
      7^2 mod 10 =  49    mod 10 = 9
      7^3 mod 10 =  343   mod 10 = 3
      7^4 mod 10 =  2401  mod 10 = 1
      7^5 mod 10 =  16807 mod 10 = 7 <- the end of the generation
                                        loop, it is seen that 7 isn't
                                        a generator.

       .--> 7 ---> 9 ---> 3 ---> 1 --. We can't generate (2,4,5,6,8)
       '------------------------------'

* 8 - Hmmm...goes out
* 9 - Hmmm...goes out
```

Well... what is going on? There is no generator! Why? The answer is simple: we haven't checked if 1,...,9 with an *(mod 10) operator generates a group. And that is in this case, because not for every element an inverse element exists. For example (5*a) mod 10 = 0 or 5, so there is no such an element b, that (5*b) mod 10 = 1.

—¿ :) And this time the solution are prime numbers (: ¡—

A. Fact 2

*If p is prime then 1,...,p-1 with *(mod p) is a group. Moreover it is a cyclic group.*

The proof of Fact 2 isn't to difficult, but there is no time and place to talk about it. It's worth saying that if n is composite number then 1,..,n-1 with *(mod n) isn't a group.

Ok, we have the group 1,...,p-1 with an *(mod p) operation and from Fact 2 we know that it's cyclic. We see that we have to round up the code size to the closest and greater prime number. To find prime numbers we could use Erastotenes algorithm, but I rather recommend the other method based on the Miller-Rabin test. All we have to do is check (for a few values a) bellow equation:

$$a^{(N-1)} \ mod \ N = 1 \tag{1}$$

If Equation 1 = true then N (odd) is probably prime number. If Equation 1 = false then N is for sure composite. (Theorem 1) It is I suppose the best know primarity test. If numbers are small it's enough to make a test for a few of them. In my prog I test for a=2,3,...,20. And it's enough for sure.

To compute $a^{(N-1)} \ mod \ N$, I suggest to use a fast and simple algorithm:

1) Modular exponentiation algorithm:

```
|-------------------------------------------------------------------------.
|    * input: a, b, n (a^b mod n)                                         |
|                                                                         |
|        c := 0                                                           |
|        d := 1                                                           |
|        Let <b_k,b_(k-1),...,b_0> be the binary representation of b      |
|        for i := k downto 0 do                                           |
|         begin                                                           |
|          c := 2 * c;                                                    |
|          d := (d * d) mod n                                             |
|          if b_i = 1 then                                                |
|           begin                                                         |
|            c := c + 1;                                                  |
|            d := (d * a) mod n                                           |
|           end;                                                          |
|         end;                                                            |
|        return d;                                                        |
|                                                                         |
|    * output: d = (a^b mod n)                                            |
'-------------------------------------------------------------------------'
```

Ok, we have the group and the operation. Let's find a generator. I don't know any tricky methods, so let's find it using "brute force" method.

But how? ;)

If g is a generator it must generate itself. So, $g^1 \ mod \ p = g$ but also $g^p \ mod \ p = g$. Let's notice that there is no such b where $g^b \ mod \ p = g$ and $b < p$. To find the generator we use the following algorithm:

```
.--------------------.
| Finding Generator |
|-------------------------------------------------------------------------.
|    * input: p - divisor from group definition                           |
|                                                                         |
|                          a := 1;                                        |
|                          Label1:                                        |
|                           a := a + 1;                                   |
|                           i := 2;                                       |
|                           while ((aˆi mod p) <> a) do i := i + 1;       |
|                          if i<>p goto label1                            |
|                                                                         |
|    * output: a - generator                                              |
'-------------------------------------------------------------------------'
```

Let G = group (1,...,5002,*(mod 5003)).

In this way we find the first generator. For the group G it is 2. We can however find the others generators. The number of them can be quite large. For the group G it is 2400. We can compute the other generators using:

B. Fact 3

g - generator, if a and p-1 are coprime then g^a is also a generator

The GCD function is very easy to code (particulary in asm). We can use:

C. Fact 4

If a¿=0 and b¿0 then GCD(a,b) = GCD(b, a mod b) —

Look on below function:

D. GCD algorithm

```
|-------------------------------------------------------------------------.
|    * input: a,b a>b                                                     |
|                                                                         |
|      Function GCD(a,b)                                                  |
|        If b = 0                                                         |
|         then return a                                                   |
|         else return GCD(a, a mod b);                                    |
|                                                                         |
|    * output: a = GCD(a,b)                                               |
'-------------------------------------------------------------------------'
```

...very, very easy and simple :)

I recommend finding various generators. For the group G mentioned above it is easy to find large generator which provides good ”random” numbers. For example: 1835 which generates 1835, 206, 2785, 2412, 3368, ...,1.

Ok, now we have all we wanted: group and generator. Let’s see a pseudocode which decodes all bytes of indexes from 1 to p-1.

E. Decoding algorithms

```
|-------------------------------------------------------------------------.
|     * input: p - divisor from group definition                          |
|              g - generator                                              |
|                                                                         |
|                       i := 1;                                           |
|                       Label1:                                           |
|                        Decode(i);                                       |
|                        i := (i * g) mod p;                              |
|                       if i <> 1 goto Label1;                            |
|                                                                         |
| or something like this:                                                 |
|                                                                         |
|     * input: p - divisor from group definition                          |
|              g - generator                                              |
|                                                                         |
|                       r := random(p-1);                                 |
|                       i := g^r                                          |
|                       Label1:                                           |
|                        Decode(i);                                       |
|                        i := (i * g) mod p;                              |
|                       if i <> g^r goto Label1;                          |
|                                                                         |
`-------------------------------------------------------------------------'
```

There is a problem - how to mutate mul and div instructions?

```
- mul
  We can write every number as a linear combination of 2^x.
  If we use such a representation we can apply shl, sal,...
  For example:
   a * 81 = (a shl 6)+(a shl 4)+a :)

- div
  Think by yourself, because I don’t know any smart method :(
```

That’s all. What’s left to do is just coding which means really hard work :)

VI. About the CGAGF program

I’m still working on the polimorphic engine using algorithms described above but I’m not gonna finish it soon so I attached a little program which contains coded algorithms. When using this program you can see that every group has many generators and they generate ”random” numbers between 1 and p-1. You can compile it with Delphi or FPC (without classes and components).

VII. Final Words

I hope you find this article interesting. Thank You for readnig :)))

I would like to thank some people:

Ish - for his help :)
Tomasz - He knows for what :)

and big thanks for The Mental Driller :)))

VIII. Bibliography

[1] Neal Koblitz "ALGEBRAIC ASPECTS OF CRYPTOGRAPHY"
[2] H.Cormen, Charles E. Leiseron, Ronald L. Rivest
"INTRODUCTION TO ALGORITHMS"
[3] My notes :)

```
/====                                                              ====\
|   When I used to learn algebra I never knew where it would lead me.   |
\====                                                              ====/
```


```
                                  /==================================\
                                  |   WhiteHead, whead@box43.gnet.pl   |
                                  | Straight Edge - better way of Life |
                                  \==================================/
```