TRABAJO DE INVESTIGACION E IMPLEMENTACION DE ALGORITMOS DE EXPONENCIACIÓN

Link Github:
https://github.com/xiop-torres/XiomaraPumaAA/tree/main/TrabajoDeInvestigacionExponenciacionModular

Integrantes:
- Xiomara Leonor Puma Torres
- - Resto Chino
- -Teorema Farmet
- Santiago Alonso San Roman Olazo
- Exponenciación Modular Rapida
- Exponenciación Modular Binaria

- Resumen

El pequeño teorema de Fermat se ha utilizado históricamente para analizar la descomposición en producto de factores primos de ciertos enteros.El teorema del resto chino tiene importantes aplicaciones en criptografía , en especial para reducir operaciones con números enormes mediante el paso a congruencias.

- Introducción

Una de las operaciones aritméticas más importantes para la criptografía de clave pública es la exponenciaciòn. Este documento analiza los métodos para calcular la G^e exponencial, donde la base g es un elemento de un grupo finito G y el exponente e es un número entero no negativo

- Contenido Teórico

A. Teorema del resto chino

   a. *Definición*:
   Dado dos números enteros $n_1, n_2 \dots, n_r$(r > 2) que cumplen (n_i, n_j)=1 para $i \neq j$, los enteros $a_1, a_2 \dots a_r$ existe x tal que

$$x \equiv a_1 (\text{mod } n_1)$$
$$x \equiv a_2 (\text{mod } n_1)$$
$$\dots$$
$$x \equiv a_r (\text{mod } n_r)$$

   Tiene una solución única módulo n = $n_1, n_2 \dots, n_m$

x = $a_1 N_1 y_1 + a_2 N_2 y_2 + \ldots + a_n N_n y_n$

donde $N_K = \frac{n}{n_k}$ y además $N_k y_k \equiv 1(mod\ n_k)$

b. Seguimiento Numérico

Ejemplo :

$x \equiv 1(\text{mod } 7)$

$x \equiv 2(\text{mod } 8)$

$x \equiv 3(\text{mod } 9)$

(1) P = 7 x 8 x 9 = 504

(2)

$p_1 = 504/\ 7 = 72$

$p_2 = 504/\ 8 = 63$

$p_3 = 504/\ 9 = 56$

(3) $q_i P_i \equiv 1(mod\ n_k)$

$q_1 72 \equiv 1(mod\ 7)$

$q_1 2 \equiv 1(mod\ 7)$

$q_1 \equiv 2^{-1}.1(mod\ 7)$

$q_1 = 6$

$q_2 63 \equiv 1(mod\ 8)$

$q_2 7 \equiv 1(mod\ 8)$

$q_2 \equiv 1^{-1}.1(mod\ 8)$

$q_2 = 1$

$q_3 56 \equiv 1(mod\ 9)$

$q_3 2 \equiv 1(mod\ 9)$

$q_3 \equiv 2^{-1}.1(mod\ 9)$

$q_3 = 5$

(4) X = 498 + 504 k

d. Implementación en c++

```
#include <NTL/ZZ.h>
#include <iostream>
using namespace std;
using namespace NTL;
const int tam = 3;

void restoChino(ZZ *ai, ZZ *pi ){
    ZZ Pi[tam];
    ZZ qi[tam];
    ZZ x0[tam];
    ZZ X = ZZ(0);
    ZZ P= ZZ(1);
    if(sonPrimos(pi,ZZ(tam))==1){
        //multilicar todos sus pi para la P resultante
        for(int i=0;i<tam ;i++){
            P=P*pi[i];
        }
        //Pi = P/pi
        for(int i=0; i <tam ;i++){
            Pi[i]=P/pi[i];
            qi[i]= inversa(Pi[i],pi[i]);
            x0[i]=ai[i]*Pi[i]*qi[i];
            x0[i]=mod(x0[i],P);
            X=X+x0[i];
        }
        X=mod(X,P);
        cout << "La solucion del sistema de ecuaciones es: " << endl;
        cout << "X = " << X << " + "<< P <<"k" << endl;
       }
}
int main()
{
    cout << "Teorema resto chino: " << endl;
    ZZ ai[tam] = {ZZ(12),ZZ(19),ZZ(7)};
    ZZ pi[tam] = {ZZ(25),ZZ(26),ZZ(27)};
    restoChino(ai, pi);
}
```

B. Exponenciación modular rápida

a. Definición:

Usando las reglas de la exponenciación modular, por este medio se puede calcular rápidamente cosas como *7^256 mod 13,*

*A^B mod C = (A * A) mod C = ((A mod C) * (A mod C)) mod C*

siempre y cuando B sea una potencia de 2.

b. Seguimiento numérico:

7^1 mod 13 = 7

*7^2 mod 13 = (7^1 *7^1) mod 13 = (7^1 mod 13 * 7^1 mod 13) mod 13*

*7^2 mod 13 = (7 *7) mod 13 = 49 mod 13 = 10*

7^2 mod 13 = 10

*7^4 mod 13 = (7^2 *7^2) mod 13 = (7^2 mod 13 * 7^2 mod 13) mod 13*
*7^4 mod 13 = (10 * 10) mod 13 = 100 mod 13 = 9*
7^4 mod 13 = 9

*7^8 mod 13 = (7^4 * 7^4) mod 13 = (7^4 mod 13 * 7^4 mod 13) mod 13*
*7^8 mod 13 = (9 * 9) mod 13 = 81 mod 13 = 3*
7^8 mod 13 = 3

*7^256 mod 13 = (7^128 * 7^128) mod 13 = (7^128 mod 13 * 7^128 mod 13) mod 13*
*7^256 mod 13 = (3 * 3) mod 13 = 9 mod 13 = 9*
7^256 mod 13 = 9

c. pseudo-algoritmo

```
incluir NTL/ZZ

Declarar funcion ZZ expomod(ZZ a, ZZ b, ZZ c)
   si A = 0,
      retorna 0
   si B = 0
      retorna 1
   ZZ operando (en este caso “y”)
   si B módulo 2 = 0
      operando = expomod(a, b/2, c)
      operando = (operando*operando) módulo c
   sino
      operando = a modulo c
      operando = (operando * expomod(a, b-1, c) modulo c) modulo c
```

d. implementación c++

```
#include <iostream>
#include <NTL/ZZ.h>
using namespace NTL;
using namespace std;


ZZ modExp(ZZ A, ZZ B, ZZ C)
{
   if (A == ZZ(0)) {
      return ZZ(0);
   }
   if (B == ZZ(0)) {
      return ZZ(1);
   }
```

```
    ZZ y;
    if (B % 2 == ZZ(0)) {
        y = modExp(A, B / 2, C);
        y = (y * y) % C;
    }
    else {
        y = A % C;
        y = (y * modExp(A, B - 1, C) % C) % C;
    }
    return (ZZ)((y + C) % C);
}

int main()
{
    ZZ A = ZZ(7), B = ZZ(256), C = ZZ(13);
    cout << A << "^"<< B << " mod" << C
         << " = "<< modExp(A, B, C);
    return 0;
```

C. Exponenciación modular binaria

a. Definición:
   Del mismo modo que la exponenciación modular rápida se puede hacer de forma rápida el cálculo sin requerir que B sea una potencia de 2.
   Para ello se divide este mismo en potencias de 2 pasándolo a binario.

5^117 mod19 = 1110101

b. Seguimiento numérico:

$$117 = (2^0+2^2+2^4+2^5+2^6)$$
$$117 = 1+4+16+32+64$$

5^1 mod 19 = 5
*5^2 mod 19 = (5^1 * 5^1) mod 19 = (5^1 mod 19 * 5^1 mod 19) mod 19*
*5^2 mod 19 = (5 * 5) mod 19 = 25 mod 19*
5^2 mod 19 = 6
*5^4 mod 19 = (5^2 * 5^2) mod 19 = (5^2 mod 19 * 5^2 mod 19) mod 19*
*5^4 mod 19 = (6 * 6) mod 19 = 36 mod 19*
5^4 mod 19 = 17
*5^8 mod 19 = (5^4 * 5^4) mod 19 = (5^4 mod 19 * 5^4 mod 19) mod 19*
*5^8 mod 19 = (17 * 17) mod 19 = 289 mod 19*
5^8 mod 19 = 4
*5^16 mod 19 = (5^8 * 5^8) mod 19 = (5^8 mod 19 * 5^8 mod 19) mod 19*
*5^16 mod 19 = (4 * 4) mod 19 = 16 mod 19*
5^16 mod 19 = 16
*5^32 mod 19 = (5^16 * 5^16) mod 19 = (5^16 mod 19 * 5^16 mod 19) mod 19*
*5^32 mod 19 = (16 * 16) mod 19 = 256 mod 19*
5^32 mod 19 = 9
*5^64 mod 19 = (5^32 * 5^32) mod 19 = (5^32 mod 19 * 5^32 mod 19) mod 19*
*5^64 mod 19 = (9 * 9) mod 19 = 81 mod 19*
5^64 mod 19 = 5

*5^117 mod 19 = (5^1 * 5^4 * 5^16 * 5^32 * 5^64) mod 19*

*5^117 mod 19 = (5^1 mod 19 * 5^4 mod 19 * 5^16 mod 19 * 5^32 mod 19 * 5^64 mod 19) mod 19*
*5^117 mod 19 = (5 * 17 * 16 * 9 * 5) mod 19*
5^117 mod 19 = 61200 mod 19 = 1
5^117 mod 19 = 1

c. pseudo-algoritmo

D. Teorema pequeño de Fermat

a. *Definición*:

Para establecer el teorema “pequeño” de Fermat , observemos que

$$(a \cdot 1)\,(a \cdot 2)\cdots(a \cdot (p-1)) = a^{p-1}(1 \cdot 2\cdots(p-1))$$

Pero podemos probar que el conjunto {a · 1, a · 2,⋯ , a ·(p − 1)} es una permutación, módulo p, del conjunto {1, 2,..., p − 1}. Luego, si mcd(a, p) = 1, cancelando tendríamos a p−1 ≡ 1 (mod p.)

Sea p primo y a ∈ Z. Si mcd(a, p) = 1 entonces

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

si p|a entonces,

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

b. Seguimiento Numérico

a^{p-1}≡1 mod p	$6^{10} mod\ 11 \ = \ 1$
a^{p}≡a mod p	$3^{12} mod\ 11$ $(3^{11} mod\ 11)($ $3^{1} mod\ 11) \ = \ (3x3) mod\ 11 \ = \ 9$

c. Implementación c++

```
ZZ fermat_peque(ZZ pos,ZZ e,ZZ p){
    if(PrimeTest(p,10)&& euclides(pos,p)==1){
        if((p-1) == e)
            return to_ZZ(1);
        if(p<e){
            return expomod(pos,e,p);
        }
    }
}
```

- Análisis de los algoritmos

. Comparar algoritmos por el cálculo computacional

a) Características

Operating System	Microsoft Windows 10 Home Single Language
RAM	8GB
Processor	Intel(R) Core(™) i5-7200U CPU @2.50 GHz
Software Tools	C++ CodeBlocks
Tipo de Sistema	PC basado en x64

b) Exponenciación modular

Nro BITS	Exp. Modular Rápido	Exp. Modular Binario
8	0.001 s	0.002 s
16	0.004 s	0.002 s
256	0.084 s	0.010 s
1024	0.721 s	0.560 s

c) Teorema resto chino y Teorema Pequeño de Fermat

Nro BITS	Teorema Resto Chino	Pequeño Teorema de Fermat
8	0.1 s	0s
16	0.05s	x
256	x	x
1024	x	x

- Conclusiones Generales

En conlusiòn se realizó 3 algoritmos que fueron Exponenciación Modular Rápida y los teoremas del resto Chino y Fermet

De acuerdo al análisis se llegó a la conclusión que La exponenciación modular ràpida fue el mejor segùn el número de bits.

- Referencias

[1] http://cacr.uwaterloo.ca/hac/about/chap14.pdf

[2]https://repositoriotec.tec.ac.cr/bitstream/handle/2238/6299/introducci%C3%B3n-teor%C3%ADa-n%C3%BAmeros.pdf?sequence=1&isAllowed=y

[3]https://es.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/fast-modular-exponentiation