CIENCIA DE LA COMPUTACIÓN

TRABAJO DE INVESTIGACIÓN E IMPLEMENTACIÓN DE ALGORITMOS QUE PERMITAN GENERAR NÚMEROS ALEATORIOS GRANDES Y ALGORITMOS QUE PRUEBEN LA PRIMALIDAD DE UN NÚMERO GRANDE

ÁLGEBRA ABSTRACTA

DANIELA CHAVEZ AGUILAR

GIULIA NAVAL FERNANDEZ

PAOLO DELGADO VIDAL

RODRIGO TORRES SOTOMAYOR

PABLO CARAZAS BARRIOS

AÑO 2021

RESUMEN

En el presente trabajo se muestra un análisis y comparación de diversas versiones algoritmos de generación de números aleatorios y de primalidad.

Aleatorios:

- Blum-blum-shub
- Micali-Schnorr
- RSA PRBG

Test Primos:

- Fermat
- Solovay-Strassen
- Miller-Rabin
- Criba de Eratóstenes

1. INTRODUCCIÓN

Analice cada uno de los algoritmos y determine las ventajas y desventajas de cada uno de ellos, ¿determine cuál de todos es el más eficiente?, Es decir quién converge más rápido. Cuál demora menos tiempo. Quién soporta trabajar con la mayor cantidad de bits.

CONTENIDO TEÓRICO

ALGORITMOS DE ALEATORIEDAD

a.Blum-blum-shub

Definición: el algoritmo Blum-Blum-Shub es un algoritmo generador de números pseudo-aleatorios propuesto por Lenore Blum, Manuel Blum y Michael Shub. Mediante una serie de exponenciaciones modulares y números primos de Blum (números primos congruentes a 3 módulo 4) generan una secuencia de bits.

Pseudocódigo:

ENTRADA: Tamaño de bits

SALIDA: Número Random

- ZZ p, q, n, seed, randnum
- string resultado
- p←random_blum_prime
- q←random_blum_prime
- seed←seed_generator(bits)
- n←p*q
- Mientras bits sea mayor que 0:
    - randnum ← (seed*seed)mod n
    - seed ← randnum
    - resultado ← randnum mod 2
    - bits--
- return strint2ZZ(resultado)

Seguimiento:

Randomnum	bit
2987	1
23994	0
16292	0
4793	1
31736	0
29714	0
28694	0
21771	1
Resultado = 137	Resultado = 10010001

Código C++:

```
ZZ random_numberBBS(ZZ bits){
    ZZ p,q,n,seed,nextRandNum;
    string resultado="";
    p = random_blum_prime(bits);
    q = random_blum_prime(bits);
    seed = seed_generator(bits);
    n = p*q;
    resultado += "1";
    while(bits != 1){
        nextRandNum = divi((seed*seed),n);
        seed = nextRandNum;
        resultado += Num2Str(par(nextRandNum));
        bits--;
    }
```

```
return binary2ZZ(resultado);}
```

b. Micali-Schnorr

Definición: Es más eficiente que el RSA PRBG desde [N (1-2/e) bits se generan por exponenciación por e.

Pseudo Algoritmo:

Input: n (bits

Outcome: Secuencia pseudoaleatoria de n bits

1. Setup: generar claves p, q, n, phi(n) y e. Con ellas definir N=cantidad de bits de n, k=floor(N(1-2/e)) y r=N-k.
2. Generar una semilla aleatoria de r bits para x_0
3. Por cada i entre 1 y l:
    a. $y_i = x_{i-1}^{e} \ mod \ n$
    b. $x_i = los \ r \ MSB \ de \ y_i$
    c. $z_i = los \ k \ LSB \ de \ y_i$
4. Retornar $z_1 \| z_2 \| z_3 \| \dots \| z_l$ ($\|$ $denota \ concatenación$)

Código C++

```
ZZ micali_schnorr(long N){

        Generar_claves(N);

  ZZ k=N*(1-ZZ(2)/e);

        long r=N-k;

        long l=N/k;

 vector<ZZ> y, z;

 vector<ZZ> x;
```

```
 x[0] = RandomBnd(power_ZZ(2,N)-1);

 for (long i=1; i<l; i++){

        y[i]= right2left_binary_modexp(x[i-1],e,n);

        x[i]= y[i]>>(N-r);

        z[i]= y[i]<<(N-r);

 }

 return vector2num(z);

}
```

c. RSA PRBG

Definición: El generador RSA es un generador de algoritmos criptográficamente seguro, asumiendo que el problema del RSA es intratable (romper el algoritmo).

Pseudo Algoritmo:

Input: n (bits)

Outcome: Secuencia pseudoaleatoria de n bits

1. Setup: generar claves p, q, n, phi(n) y e.
2. Generar una semilla aleatoria entre 1 y n-1 para x_0
3. Por cada i entre 1 y l:
   a. $x_i = x_{i-1}^{e} \ mod \ n$
   b. $z_i = LSB \ de \ x_i$
4. Retornar $z_1 \| z_2 \| z_3 \| \dots \| z_l$ ($\|$ *denota concatenación*)

Código c++:

```
long RSA_PRNG(long N){

Generar_claves(N);

 vector<long> y, z;

 vector<long> x [0] = divi(seed_generator(N),n);
```

```
 for (long i=1; i<N; i++){
        x[i]= rigth2left_modexp(x[i-1],e,n);
        z[i]=x[i]-((x[i]>>1)<<1);
 }
 return vector2num(z);
}
```

ALGORITMOS DE PRIMALIDAD

a. Fermat

Definición: Según el Teorema de Fermat, si un número n es primo y a es un entero entre 1 y n-1, entonces $a^{n-1} \equiv 1 \, mod \, n$. A partir de esto, tenemos que, dado un número *n* cuya primalidad está en cuestión, basta encontrar un entero *a* en dicho rango que no cumpla la igualdad, para demostrar que *n* es compuesto.

Pseudo-algoritmo:

Input: n>=3 y que sea impar, t>=1

Output: Es primo true, no es primo False

1. Por cada i en el rango 1-t:
    a. Elegir un número aleatorio entre 2 y n-2
    b. Calcular $r = a^{n-1} \, mod \, n$
    c. Si $r \neq 1$: $return \, false$
2. $return \, true$

Código c++:

```
bool Fermat_primo(ZZ n, int k){
 for (int i=0; i<k; i++){
        ZZ a=divi(conv<ZZ>(rand()),n)+2;
        if (ExpMod(a, n-1, n)!=1)
        return false;
 }
 return true;
}
```

b. Solovay-Strassen

Definición: La prueba de primalidad probabilística de Solovay-Strassen fue la primera prueba de este tipo popularizada por la advenimiento de la criptografía de clave pública, en particular el criptosistema RSA. Ya no hay ningún motivo para utilizar esta prueba, porque hay una alternativa disponible (la prueba de Miller-Rabin) que es más eficiente y siempre al menos tan correcta.

Pseudo Algoritmo:

1. Input: n>=3 y que sea impar, t>=1
2. Output: Es primo true, no es primo False
    - ZZ a, r
    - Mientras 2<=a<=n-2
    - a=número aleatorio
    - r=a^((n-1)/2) mod n
    - Si r != 1 y r != n-1, retorna falso
    - ZZ s = Jacobi(a,n)
    - Si r != s mod , retorna falso
    - Retorna true

Código c++

```
bool Solovan_Strassen(ZZ n, ZZ t)

{

   if(n>=3 &&  divi(n,conv<ZZ>(2))!=0 && t>=1)

   {

      ZZ a, r;

      while(a>=2 &&  a<=n-2){

         a=RandomBnd(); }

      r = divi(power_ZZ(a,(n-1)/2),n);
```

```
    if(r!=1 && r!=n-1) return false; //composite

    ZZ s;

    s = Jacobi(a,n);

    if(r!=divi(s,n) return false; //composite

    return true; //prime

  }

}
```

c. Miller-Rabin

Definición: es un test de primalidad, es decir, un algoritmo para determinar si un número dado es primo, similar al test de primalidad de Fermat. Su versión original fue propuesta por G. L. Miller, se trataba de un algoritmo determinista, pero basado en la no demostrada hipótesis generalizada de Riemann; Michael Oser Rabin modificó la propuesta de Miller para obtener un algoritmo probabilístico que no utiliza resultados no probados.

Pseudocódigo:

1. Input: n y t
2. Output: Si es primo true si no false
    - ZZ a, y, i, j, s, r
    - Mientras n>1
        - n se divide entre 2
        - s se aumenta en 1
    - Mientras i<=t
    - a=número aleatorio mayor a 2 y menor a n-2
    - y=a^r mod n
    - Si y != 1 y y != n - 1
        - j=1
        - Mientras j<=s-1 y y != n-1
            - y=y^y mod n
            - Si y = 1, retorna false
            - y se aumenta en 1
        - SI y != n-1, retorna false
    - Retorna true

Código C++

```
bool miller_rabin(ZZ n,long t){
    if (n<2) return false;
    if (n==2 || n==3) return true;
    //if(par(n)) return false;//ahorrar trabajo si es par
    ZZ a,y,i,j,s,r;
    s=0;
    r=n-1;
    while(r>1 && par(r)){
        r>>=1;
        s++;
    }
    for(i=1;i<=t;i++){
        a=ZZ(2)+divi(seed_generator(bitz), n-3);
        y=right2left_binary_modexp(a,r,n);
        if(y!=1 && y!=n-1)
        {
            j=1;
            while(j<=s-1 && y!=n-1)
            {
                y=divi(y*y,n);
                if(y==1)return false;
                j++;
            }
            if(y!=n-1)return false;
        }
```

```
    }
    return true;
}
```

d. Criba de Eratóstenes

Definición: es un algoritmo que permite hallar todos los números primos menores que un número natural dado. Se forma una tabla con todos los números naturales comprendidos entre 2 y n, y se van tachando los números que no son primos de la siguiente manera: Comenzando por el 2, se tachan todos sus múltiplos; comenzando de nuevo, cuando se encuentra un número entero que no ha sido tachado, ese número es declarado primo, y se procede a tachar todos sus múltiplos, así sucesivamente. El proceso termina cuando el cuadrado del siguiente número confirmado como primo es mayor que n.

Código C++:

```
vector<bool> isPrime;
vector<long long> primes3;
void criba3(long long n) {
    primes3.push_back(2);
    primes3.push_back(3);
    for (long long  i=5; i<n; i=i+2)  {if (divi(i,3)!=0) primes3.push_back(i);}
    for (long long  i=2; (primes[i])*(primes[i])<n; i++) {
        for (long long  k=i+7; k<primes3.size(); k++) {if (divi(primes3[k],i)== 0){
primes3.erase(primes3.begin()+k-1);primes3.resize(primes3.size());primes3.shrink_to_fit();}
        }
    }
}
```

ANÁLISIS DE ALGORITMOS

Para los algoritmos utilizamos: Codeblocks usando la librería NTL para probar con los distintos números de bits.

Sistema Operativo: Windows 10

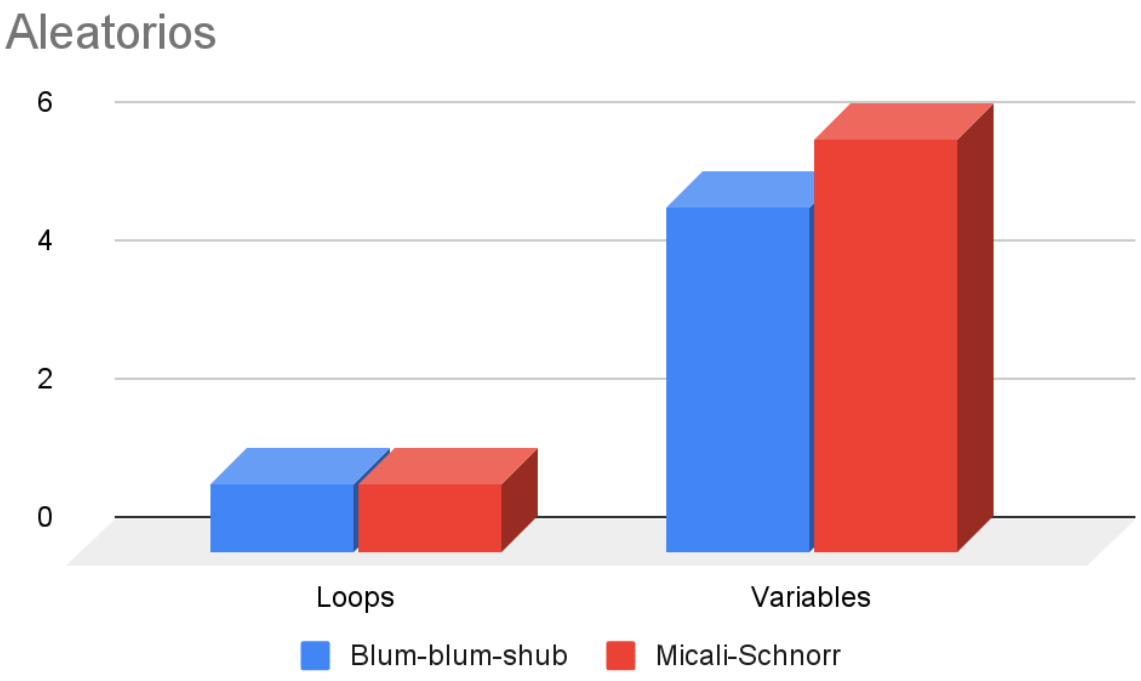
Algoritmos de aleatoriedad

	Blum-blum-shub			Micali-Schnorr	
Bits	Número #1	Número #2		Número #1	Número #2
8	137	231		171	182
16	50525	33609		49914	57606
32	2251637947	761555861		4209612127	3542448294
64	11185338181189285525	12736617726848423495		97975138219290395567	95359441484828613307
128	338566200913088282816198675824456666445	32704098051558049391055971 4654050581653		23700130296327500888051390 9949532676811	61281701206208644
256	771964585164050615269171948464190843311069846080389270850272816720843956578 69	82304241519319681463307098 0009033377427 4033686330000 1421277095875 554348778865		82599590510124009456390007 4827205325648 9102063942394 11161140197098 04182312591	27743133640186700
512	937450714132612182764590800574412697837967595116743002956201753550876010812644062090690304574458414714475433513419297976024050097709674592474091457264 1212	13363872615139082476408901 77252732721187 2997357693591 6887497957834 6455245624641 3252185007653 5207135892387 9926373947523 51717981132769 8248471868406 3253576487		74305910992485723771247912 4704667440576 9250668221169 9470043174585 0766095344692 1473684907501 3628766838371 9122062569368 3430664232774 3825362935113 80628245016	05889674383796837
1024	169635026845994978752912352416	15796399612451229288980666	Micali-Schnorr	13125301891818416765602493	42581742696225175195172317193916

	932823746548105 971590145416992 703776670171977 333085596363277 893739420647806 245632012952276 357042607844580 226314373766962 738565248106086 579521151365651 907883650639997 711247615630207 164918006697941 344827381570619 542889625797798 474920782219783 544361192274192 965692556694147 754606970	3693901673209 4923050816927 4472025592186 3134168669243 0132224140064 7582927181317 6375031294265 2750336428667 2077720103416 55119020437052 1585684373416 0691723279694 17321187666057 5393910250666 89954083655611 6957413750397 6777148446624 3846862055028 73085584311901 1297107695914 7916103177884 096567		6170653250261 4722744307219 7892984311190 7640782238993 4360696280605 7366952405637 5441609872964 6046538858437 5373720763700 8136053495172 0502637138413 9681649008421 7374012172382 3713141522114 5252330325801 3593637940338 6598996723827 9786468482629 1440791070809 2824259500144 5155051841398 3621366461	
2048	168471953358133 315829067843526 322310001704885 725881689239905 907414566388223 755142931817670 315641610736177 724821887638386 392324408665900 241732339155549 518895767116844 753469041333937 109866289897291 074056630039933 598011522711745 351173469765606 945948941928229 154387956664778 636075742608263 724234125260252 691148806559459 761473660797548 001945157544400 325288682790942 143071682514576 366534257356045 665883824724239 334067856497756 059179405873470	2375873548584 5561012527006 1942051695837 7680020339218 4765668980410 8995214680934 82460927711976 11456061388752 1929434761851 7286541786428 91531106208521 5471410607349 5184357051219 8374172643084 1898649398743 67249211879398 5325887908918 7139487834225 0986298364781 7826279057800 2172953064091 0681682790224 3343338365108 4750676007060 5314100350606 5953087576219 1661516772128 3135961843868 2421398758202		709714167674667 951069858798386 691841646488506 485562859562708	

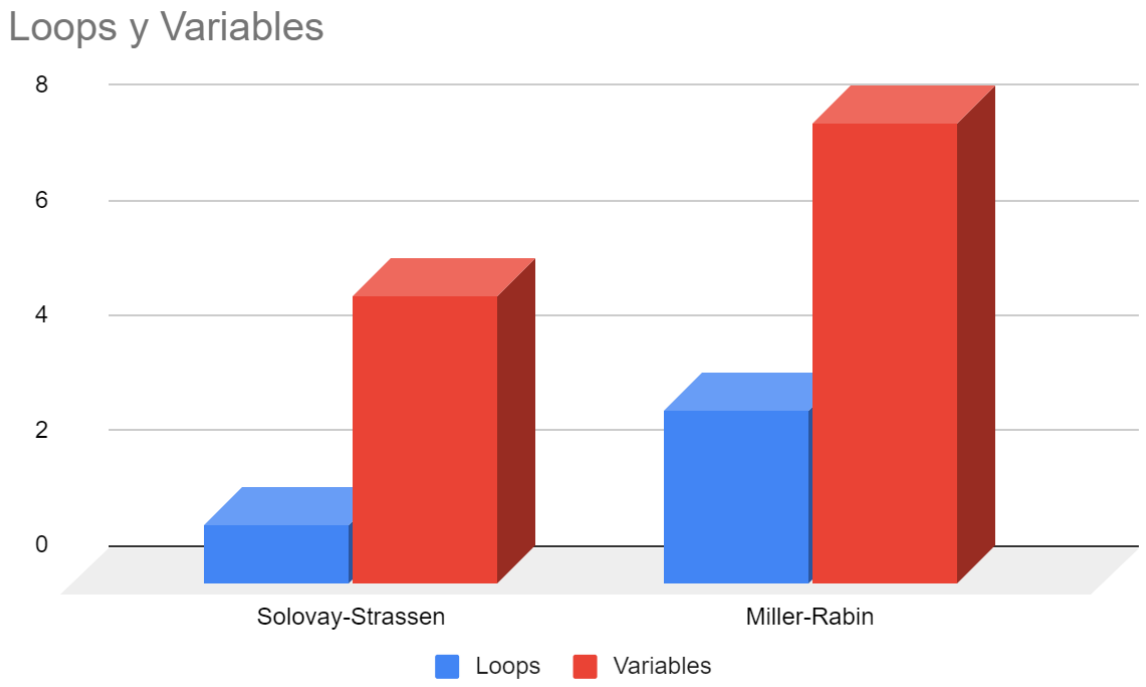
	140267174899879764608517485894573261850792147454835238065582007852959980972582320485362876531284695748530981332272533523776704369512722026657368603239081241812506005712153904807644 44	7370719958802593013359889737275599385151947073462079294704206232867558052538387770347168893459567633974534656417487469820197997761843442002390519433604980050824404412507108544584662835913449487292364113257006768943108362795473913428794			
Promedio	750562869,7	253863233,7		1403220737	1180835361

	Blum-blum-shub	Micali-Schnorr
Loops	1	1
Variables	5	6



PRIMALIDAD

	Solovay-Strassen	Miller-Rabin
Loops	1	3
Variables	5	8



CONCLUSIONES GENERALES

Como conclusión obtuvimos que el algoritmo más eficiente de aleatoriedad es el Blum-Blum-Shub ya que después de hacerle las pruebas respectivas esta se encuentra dentro del promedio. Y para el algoritmo de primalidad el más eficiente es el test de Miller-Rabin.

REFERENCIAS

Generación de números aleatorios:

Handbook of Applied Cryptography, Menezes, Oorschot, Vanstone. CRC Press, New York, fifth edition (2001). http://www.cacr.math.uwaterloo.ca/hac/

Capítulo 4: Public key parameters. Random search for probable primes. Página 145. http://www.cacr.math.uwaterloo.ca/hac/about/chap4.pdf

Capítulo 5: Pseudorandom Bits and Sequences http://cacr.uwaterloo.ca/hac/about/chap5.pdf

MARTON, Kinga; SUCIU, Alin; IGNAT, Iosif. (2010) Randomness in digital cryptography: A survey. Romanian Journal of Information Science and Technology, , vol. 13, no 3, p. 219-240. http://romjist.ro/content/pdf/kmarton.pdf

STIPČEVIĆ, Mario; KOÇ, Çetin Kaya. True random number generators. En Open Problems in Mathematics and Computational Science. Springer, Cham, 2014. p. 275-315. http://cetinkayakoc.net/docs/b08.pdf

Generación de primos:

Breve Reseña sobre la Hipótesis de Riemann, Primalidad y el Algoritmo AKS http://www.criptored.upm.es/guiateoria/gt_m117j.htm

Probabilidad, Números Primos y Factorización de Enteros. Implementaciones en Java y VBA para Excel. Revista digital Matemática, Educación e Internet https://tecdigital.tec.ac.cr/revistamatematica/HERRAmInternet/v8n2-DIC007/Probabilidad_Primos_Factorizacion.pd

Los enigmáticos números primos https://www.yumpu.com/es/document/read/14281137/los-enigmaticos-numeros-primoscinvesta