**Universidad Central de Venezuela**
**Facultad de Ciencias**
**Escuela de Computación**

***Lecturas en Ciencias de la Computación***
***ISSN 1316-6239***

# Generadores de números seudo aleatorios y criptografía

Prof. R.A. Pastoriza

**RT 2008-07**

Centro de Investigación de Operaciones y Modelos Matemáticos Aplicados
CIOMMA
Caracas, Noviembre, 2008.

RT-2008-7
Lecturas en Ciencias de la Computación
ISSN 1316-6239

# Generadores de números seudo aleatorios y criptografía

R. A. Pastoriza*
rpastoriza@cantv.net

noviembre 2008

**Abstract**

We present an experimental study of the quantitative properties of some know and new PRNG. Two new PRNG porfiadoUL_CR y FF_CR are presented as cryptography resistant.
Some preliminary experiments shows very good pseudo random properties.

**Resumen**

Se presenta un estudio experimental de las propiedades cuantitativas de algunos generadores conocidos y otros nuevos introducidos en este estudio. Se introducen dos generadores de números seudo aleatorios considerados criptográficamente resistentes.
Las pruebas empíricas realizadas permiten concluir que porfiadoUL_CR y FF_CR los algoritmos de generación de números seudoaleatorios propuestos poseen buenas propiedades estadísticas.

**Palabras clave:** pseudorandom generator, criptography, random

*CIOMMA. .Centro de Investigación de Operaciones y Modelos Matemáticos Aplicados, Escuela de Computación, Facultad de Ciencias, Universidad central de Venezuela, Los Chaguaramos, Apartado 47002, Caracas, 1041-A, Venezuela.

Resumen
1.- Introducción
2.- Aleatoriedad
2.1.- Pruebas empíricas de aleatoriedad
2.2.- GNSA no tan buenos
2.3.- GNSA que pasan Tuftest
3.- GNSA en Criptografía
3.1.- porfiadoUL_CR
3.2.- Kiss_me
3.3.- girar_me
3.4.- dinámico_CR
3.5.- isaac_CR
3.6.- reticulado_CR
3.7.- FF_CR
4. Conclusiones
Referencias
apéndice 1
GNSA no tan buenos
apéndice 2
GNSA que pasan las pruebas de Tuftest
apéndice 3
GNSA criptográficamente resistentes

## 1.- Introducción

En un trabajo previo el autor, [13] Pastoriza, presentó ⊕P⊕ un algoritmo aleatorio de cifrado por bloques. Este requiere disponer, como se debe esperar, de un par de generadores de números seudoaleatorios criptográficamente seguros, dado que se debe poder cifrar un cierto texto y descifrarlo correctamente.

En este trabajo se busca determinar cuales - GNSA - generadores de números seudo aleatorios son adecuados para su empleo en criptografía, dado que hay poca cantidad de ellos reportados para este fin.

Cuando se habla de GNSA para su uso criptográfico, dada su naturaleza algorítmica los valores producidos son predecibles, si se conoce el estado inicial, y el algoritmo de generación también, como suele ser el caso.

En primer lugar los números generados por los GNSA para su uso criptográfico deben ser impredecibles. O sea que tenga buenas propiedades estadísticas, los números deben parecer ser uniformes e independientes.

Si se desconoce la semilla de un GNSA el siguiente número generado debe ser impredecible aun si se conocen los valores generados previamente - forward unpredictability -, incluso conociendo el algoritmo de generación. O sea, conocido una salida/output del GNSA no se debe poder inferir cuales son los siguientes números que se van a generar.

Por supuesto también se requiere que exista impredectibilidad hacia atrás; esto es si se conoce un cierto output del GNSA debe ser imposible inferir que semilla y/o estado del generador lo produjo - backward unpredictability -.

Estas condiciones a su vez indican que la o las semillas de un GNSA en su uso criptográfico se deben mantener secretas. Y la misma semilla debe ser impredecible.

Para el diseño de los algoritmos de generación de números seudo aleatorios se han seguido algunas de las recomendaciones que aparecen tanto en los trabajos de [1] Baker y Kelsey como en [4] Kelsey, Schneier, Wagner y Hall.

En esencia se reducen a:

1.- Basar el GNSA en algo fuerte; resistente a ataques.
2.- Asegurarse que todo el estado del GNSA cambie en el tiempo.
3.- Hacer un resembrado de las semillas del GNSA.
4.- Hacerlo resistente al análisis retrospectivo - backtraking -.
5.- Hacerlo resistente a los ataques con texto natural elegido.

6.- Recuperarse de compromisos rápidamente.

En adición se sugiere incluir en el código pruebas de “salud del GNSA”. Esto es una comprobación del buen y correcto funcionamiento del generador.

Se han tomado entonces algunos GNSA conocidos y otros no tanto, para reformularlos en términos de Criptografía; a la vez se han diseñado otros nuevos siguiendo las sugerencias anteriores. Cualquiera sea el caso se ha tratado de comprobar sus bondades estadísticas.

## 2.- Aleatoriedad

### 2.1.- Pruebas empíricas de aleatoriedad

El enfoque pragmático acerca de la aleatoriedad de una determinada sucesión de números pasa por poder decidir, a los fines prácticos, que no tiene apariencia aleatoria. O dicho de otro modo son aquellas sucesiones de números que se comportan como si fuesen aleatorias.

Términos vagos estos que se traducen, desde el punto de vista computacional en un conjunto de pruebas empiricas que buscan determinar la aparente falta de uniformidad y/o independencia en una cierta sucesión numérica.

La primera referencia acerca de la comprobación de la bondad estadística de números generados artificialmente se da en el trabajo de [5] Kendal y Babington-Smith del año 1938. Ahí los autores se ocupan de estudiar el comportamiento de los dígitos 0...9 en los números publicados por la Lotería Británica. Por cierto se utilizaba un dispositivo electromecánico para generar los dígitos. Introducen el test de frecuencias, serial, separación, y poker.

Años mas tarde [6] Knuth (1969) recolectó y sistematizó en su famoso texto las pruebas disponibles para ese momento. Incluye pruebas generales para estudiar la bondad de ajuste de datos aleatorios, a la distribución uniforme, como son las pruebas de Chi cuadrado y del tipo Kolmogorov-Smirnov. Como pruebas empíricas describe los test de frecuencias serial, separación, poker, coleccionista de cupones, permutación, corridas, máximo, y correlación serial. Las pruebas teóricas mencionadas se refieren a test destinados a disponer de criterios a priori de aceptación de buenos coeficientes e involucran sólo a los generadores congruenciales lineales. Hay una versión en FORTRAN en el trabajo de [16] Sabbagh (1985).

El siguiente esfuerzo en computación fue el realizado por [9] Marsaglia quien publica en 1996 como resultado de 30 años de trabajo en el tema una batería de pruebas llamada DIEHARD, *Diehard Battery of Tests of Randomness,* que fue ampliamente aceptada, difundida, y usada.. Incluye las siguientes 15 pruebas:

1 Birthday Spacings
2 Overlapping Permutations
3 Ranks of 31x31 and 32x32 matrices
4 Ranks of 6x8 Matrices
5 Monkey Tests on 20-bit Words
6 Monkey Tests OPSO,OQSO,DNA
7 Count the 1`s in a Stream of Bytes
8 Count the 1`s in Specific Bytes
9 Parking Lot Test
10 Minimum Distance Test

11 Random Spheres Test
12 The Sqeeze Test
13 Overlapping Sums Test
14 Runs Test
15 The Craps Test

Con esta batería se prueban cada vez una secuencia con alrededor de 11Mb de enteros sin signo de 32bits, bien sea con todas las pruebas o con las que se seleccionan.

En el 2001 [15] Rukhin y otros autores del NIST - National Institute of Standards and Technology -, publican un extenso reporte acerca de pruebas estadísticas destinadas a ser usadas en criptografía; con énfasis en test de bits. Esto en razón a que los GNSA usados para este fin tienen requerimientos más fuertes que los empleados en simulación. En particular existe la exigencia que los números generados deben ser impredecibles aun si se desconocen las entradas.

Incluyen en su batería las siguientes pruebas:

1 Frequency (Monobit) Test
2 Frequency Test within a Block
3 Runs Test
4 Test for the Longest Run of Ones in a Block
5 Binary Matrix Rank Test
6 Discrete Fourier Transform (Spectral) Test
7 Non-overlapping Template Matching Test
8 Overlapping Template Matching Test
9 Maurer's "Universal Statistical" Test
10 Lempel-Ziv Compression Test
11 Linear Complexity Test
12 Serial Test
13 Approximate Entropy Test
14 Cumulative Sums (Cusum) Test
15 Random Excursions Test
16 Random Excursions Variant Test

El trabajo continuo de Marsaglia en el tema conduce en el 2006 a un artículo [12 ] con Wai Wan Tsang llamado *Some difficult-to-pass tests of randomnes.* En este se presentan tres pruebas *Tuftest* que según sus propias palabras son más exigentes que toda la batería Diehard.

Las pruebas que se incluyen son la *prueba del MCD*, del *gorila*, y la del *cumpleaños*.

La *prueba del MCD* - máximo común divisor - se puede conceptualizar así en los términos de Marsaglia. En el algoritmo de Euclides para encontrar el máximo común divisor de dos

números enteros aparecen tres cantidades significativas. (1) el número de iteraciones $k$ que se requieren para calcular el MCD, (2) una secuencia de longitud variable de enteros con los cocientes parciales, y (3) el MCD.

La prueba consiste en elegir pares de enteros al azar y construir tres listas (1) la del número de iteraciones $k$ que son variables aleatorias independientes igualmente distribuidas – iid -, (2) la lista de los cocientes parciales cuyos valores no son iid, y (3) los *mcd* que si son iid.

Marsaglia usa distribuciones empíricas de los valores $k$ y de los *mcd* para comprobar la aleatoriedad de los números seudo aleatorios producidos por generadores. La disparidad más frecuente sucede en la distribución de los $k$, el número de pasos para terminar el algoritmo de Euclides.

El *test del gorila* se apoya en la idea de producir una secuencia de letras de un cierto alfabeto, para entonces estudiar la frecuencia de esas $k$-letras en la secuencia. Produce una de las pruebas más difíciles de pasar por los GNSA. Usar secuencias muy largas sirve para descubrir debilidades como las presentes en los GNSA con períodos cortos que son incapaces de producir todas las posibles $k$-uplas. Una manera posible eficiente de implantar estas ideas es tratar de contar el número de letras que no aparecen en las $k$-uplas producidas por el gorila. Mediante simulación se determinó que dicho valor es aproximadamente normal con media teórica ( 24687971) y varianza empírica 4170. Esto cuando se toma 1 bit, en una posición fija, de los enteros de 32bits generados y se usan $2^{26} + 25$ bits; el número $x$ de palabras de 26bits es aproximadamente normal con esos parámetros.

La *prueba del cumpleaños* se basa que si se toman $m$ cumpleaños aleatoriamente de un año de $n$ días, y se los ordena, el número de valores duplicados entre los espacios entre esos cumpleaños ordenados debe ser una distribución Poisson con parámetro $\lambda = m^3/(4n)$.

No hay mucho soporte teórico pero la experimentación con varios GNSA, vía simulación permitió que los autores encontraron valores de $m$ y $n$ para los cuales la distribución Poisson parece satisfactoria. Los usados en la prueba son $m = 4096$ para un año de $n = 2^{32}$ días con $\lambda = 4$.

El criterio usado para calificar si un determinado GNSA pasa una prueba es que el valor estadístico resultante $v$ que sea superior a 0.01 y menor a 0.99.

Esta batería, *Tuftest* implantada en C, es la que se emplea en este trabajo para comprobar las bondades aleatorias o no de los generadores de números seudo aleatorios – GNSA - que se exploran en este trabajo.

Mas recientemente en el 2007 [8] L'ecuyer y Simard publican una librería en C para probar empíricamente generadores de números seudo aleatorios.

### 2.2.- GNSA no tan buenos

En esta sección se reportan algunos GNSA conocidos como con mala aleatoriedad y otros, muy usados, que sin embargo no pasan las prueba de *Tuftest*.

Todos los GNSA implantados o bien proporcionan ya enteros sin signo de 32 bits, o se transforma su salida a esa magnitud. Esto obedece a que el código C de *Tuftest* requiere como entrada un GNSA que proporcione números enteros sin signo de 32 bits.

#### 2.2.1.- Randu

*Randu* es un generador conocido por su mala calidad, y para los estándares actuales es inaceptable su uso en cualquier aplicación, por simple que sea ésta. Se remonta a los años 60, cuando como parte de la librería científica de IBM la SSP - Scientific Subroutine Package - y su codificación en FORTRAN, era de amplio uso. Fue por casualidad que se descubrió su pobre desempeño, y una vez que se encontró el problema fue fácil ver porque falla.

Aquí sirve para comprobar, una vez más la calidad de *Tuftest* como examinador empírico de las bondades aleatorias de una sucesión generada por un determinado GNSA.

*Randu*:

$$x_{n+1} \equiv 65539\, x_n \pmod{2^{31}}$$

si se hace un poco de aritmética modular se encuentra que

$$x_{n+2} \equiv 6_{n+1} - 7\, x_n \pmod{2^{31}}$$

relación que señala una alta correlación entre los términos consecutivos $x_{n+2}$, $x_{n+1}$ y $x_n$.

Los valores de *Tuftest* para *Randu* inicializado en $x_0 = 45813$ son:

Prueba del cumpleaños = 1.0
Prueba del algoritmo de Euclides
Pasos al gcd =1.0
distribución de los gcd =1.0
Prueba del gorila =1.0

Valores estos que indican la falta de aleatoriedad de las sucesiones producidas por Randu. Otras inicializaciones también producen resultados similares.

El código en C y los resultados aparecen en el apéndice 1.

### 2.2.2.- ran1

*ran1* es un generador aleatorio, congruencial multiplicativo, muy usado en simulación ya que $7^5 = 16807$ es un elemento primitivo módulo $2^{31}$-1, que es el mayor primo menor a $2^{31}$. Y esta propiedad establece un período máximo del GCL. Aparece citado en [14] Press y otros.

La recurrencia aquí es:

$$x_{n+1} \equiv 16807 x_n \ (mod\ 2^{31}-1)$$

Los resultados con *Tuftest* muestran que no tiene buenas propiedades de aleatoriedad.

El código en C y los resultados aparecen en el apéndice 1.

### 2.2.3.- cong1

*cong1* es un GNSA módulo $2^{32}$ similar al recomendado por [6] Knuth, pero con un término independiente adicional, ya que sin éste pasa la prueba espectral. Aparece también como un componente de Kiss99 en [10] Marsaglia. Sin embargo el examen con *Tuftest*, de la secuencia producida arroja malos estadísticos .

El congruencial lineal *cong1* es:

$$x_{n+1} \equiv 69069 x_n + 1234567 \ (mod\ 2^{32})$$

El código en C y los resultados aparecen en el apéndice 1.

### 2.2.4.- cong2

*cong2* es un GNSA módulo $2^{32}$ similar al recomendado por [6] Knuth, pero con un término independiente adicional, ya que sin este último pasa la prueba espectral. Sin embargo el examen con *Tuftest*, de la secuencia producida arroja malos estadísticos .

El congruencial lineal *cong2* es:

$$x_{n+1} \equiv 1099087573 x_n + 71828182845 \ (mod\ 2^{32})$$

El código en C y los resultados aparecen en el apéndice 1.

### 2.2.5.- cong3

*cong3* es un GNSA módulo $2^{32}$ similar al recomendado por [6] Knuth, pero con un término independiente adicional, ya que sin este último pasa la prueba espectral. Sin embargo el examen con *Tuftest*, de la secuencia producida arroja malos estadísticos .

El congruencial lineal *cong3* es:

$$x_{n+1} \equiv 1664525x_n + 132721788 \pmod{2^{32}}$$

El código en C y los resultados aparecen en el apéndice 1.

### 2.2.6.- cong4

*cong4* es un GNSA módulo $2^{32}$ similar al recomendado por [6] Knuth, pero con un término independiente adicional, ya que sin este último pasa la prueba espectral. Sin embargo el examen con *Tuftest*, de la secuencia producida arroja malos estadísticos .

El congruencial lineal *cong4* es:

$$x_{n+1} \equiv 1566083941x_n + 1234567 \pmod{2^{32}}$$

El código en C y los resultados aparecen en el apéndice 1.6.-.

Los cuatro generadores congruenciales lineales – GCL - anteriores son generalmente recomendados ya que, módulo $2^{32}$ , sus sucesiones pasan todas la prueba espectral, quizás una de las más exigentes dedicada a los GCL.

### 2.2.7.- SHR3

*SHR3* es un GNSA módulo $2^{32}$ que es un componente del generador *Kiss99* en [10] Marsaglia; se realizan 3 corrimientos de registro fijos y el xor con el valor anterior. Sin embargo un examen con *Tuftest*, de la secuencia producida arroja malos estadísticos .

El generador de corrimientos *SHR3* implanta consecutivamente la sucesión de operaciones:

$$jsr = jsr \oplus (jsr<<17) \pmod{2^{32}}$$
$$jsr = jsr \oplus (jsr>>13) \pmod{2^{32}}$$
$$jsr = jsr \oplus (jsr<<5) \pmod{2^{32}}$$

donde, en C, el operador binario *(x<<k)* es el corrimiento de bits hacia la izquierda del entero *x* en la magnitud indicada por el entero *k*. El $\oplus$ es el operador lógico o exclusivo *xor*.

El código en C y los resultados aparecen en el apéndice 1.

**2.2.8.- reticulado19**

Este es un generador que proviene de la teoría del caos y sistemas dinámicos no lineales mencionado en [2] González y otros. Y es ejemplo de un reticulado acoplado con propiedades tales que un sistema dinámico no periódico sea el argumento de una función no invertible.

La ecuación 19 que lo caracteriza es:

$$X_{n+1}(i) = [(a + bX_n(i-1) + cX_n(i) + dX_n(i-1))\, X_n(i) + fX_n(i-1) + gX_n(i+1) + 0.1] \pmod 1$$

*con i=-2,-1,0,1,2.*

Donde los valores reales de *a, b, c, d, f, y g* son parámetros fijos. El valor devuelto como generado es el término del medio $X_{n+1}(0)$.

Los resultados de *Tuftest* con la inicialización usada indican que se debe rechazar; sin embargo todos los estadísticos son aceptables, o muy buenos, salvo el correspondiente a la prueba del cumpleaños que tiene el valor 0.991. Otras pruebas independientes con otras inicializaciones dan resultados correctos. Esto sugiere que, independientemente de la inicialización, si por ejemplo se devuelve la suma de los tres términos centrales o una magnitud equivalente que mezcle los tres resultados es casi seguro que pase las pruebas.

El código en C y los resultados aparecen en el apéndice 1.

**2.2.9.- Kiss99_o**

Se analiza la versión original de *KISS* (Keep It Simple Stupid) o sea con la estructura y los valores originales de sus parámetros según [10] Marsaglia. Se trata de un generador compuesto que sucesivamente usa un congruencial lineal, otro de corrimiento de registros y la concatenación de dos multiplicativos, de 16 bits, con acarreo, para finalmente componer el resultado como $(CONG \oplus MWC) + SHR3$.

*KISS* como tal es la siguiente relación:

$$Kiss = (CONG \oplus MWC) + SHR3$$

donde:

*CONG: x ≡ 69069*x+ 1234567 (mod 2**32)*

*MWC: zw = ((z<<16)+ w)*
*z = 36969*(z&65535)+(z>>16)*
*w = 18000*(w&65535)+(w>>16)*

*SHR3:(jsr^=(jsr<<17), jsr^=(jsr>>13), jsr^=(jsr<<5))*

Este último *SHR3* es el generador analizado en el punto **2.2.6.-** y *CONG* es *cong1* del punto **2.2.3.-**.

Sorprende que por muy poco no logre estadísticos aceptables, ver apéndice 1.; sin embargo otras inicializaciones si resultan con estadísticos aceptables.

El código en C y los resultados aparecen en el apéndice 1.

## 2.3.- GNSA que pasan las pruebas de *Tuftest*

### 2.3.1.- porfiado

Este es un generador representante de una clase de generadores basados en transformaciones en números reales introducida y estudiada por [7] Lavieri en 1984.

Hemos tomado para revisar la versión 4 de un algoritmo porfiado en dos dimensiones, los valores empleados son todos reales, su descripción es:

Dados $x_0 = A$ *con* $A \in [100,200]$; $y_0 = B$ *con* $B \in [100,200]$; $0<F_0<1$; $0<G_0<1$;

Generar recursivamente:

$$x_{n+1} = A/(x_n+F_n)$$
$$y_{n+1} = B/(y_n+G_n)$$

$$F_{n+1} = DEC(z*y_n)$$
$$G_{n+1} = DEC(z*x_n)$$

donde *DEC(u)* indica la parte decimal del argumento *u*.

*z = 32 si el computador es binario*
*z = 100 si el computador es decimal*

La salida es $F_{n+1}$, $G_{n+1}$.

En la primera prueba *Tuftest* devolviendo el valor $F_{n+1}$ resultaron estadísticos aceptables.

El código en C y los resultados aparecen en el apéndice 2.

### 2.3.2.- Kiss99_b

Es otra versión de *Kiss99*; difiere de la original de aquí en mas llamada *Kiss99_o* porque se han cambiado el GCL, tomándose en su lugar el *Cong4* anteriormente citado.

$$x = 1566083941*x + 1234567(mod2^{32})$$

Se cambió también el *SHR3* original por el *SHR3 (16,-7,11)* con uno de los desplazamientos sugeridos en [14] Press y otros, más términos independientes:

```
jsr_b ^= (jsr_b<<16) + 123;
jsr_b ^= (jsr_b>>7)  + 456;
jsr_b ^= (jsr_b<<11) + 789;
```

El *MWC* se conserva igual.

De manera similar *Kiss99_b* es el valor *((MWC^CONG)+SHR3)*.

En la primera prueba con *Tuftest* devolviendo el valor anterior resultaron estadísticos aceptables.

El código en C y los resultados aparecen en el apéndice 2.

### 2.3.3.- girar_dec

*girar_dec* es un GNSA que se apoya en girar, el entrecruzamiento de un corrimiento de registros con la suma de otro, mezclando las operaciones *xor* y negación, más un término independiente. La composición de 4 enteros UL produce el resultado final de girar. Cuando se incluye la decimación se lo renombra *girar_dec*. Todos ellos trabajan con enteros grandes sin signo, UL en la notación de C.

*girar* es el generador primario siguiente:

```
entrada:
    a1,a2,a3,a4
calcular
        a1 = ~(a1^(a1<<5) + a2 + 1234567);
        a2 = a2^(a2>>13) + a1 + 76543;

        a3 = a3^(a3<<6)+ a4 + 314151;
        a4 = ~(a4^(a4>>7)+ a3 + 349765);

        z = (a1^a3) + (a2^a4);
salida:
    z
```

En la primera prueba con *Tuftest* resultaron estadísticos aceptables.

El código en C de *girar_dec* y los resultados aparecen en el apéndice 2.

### 2.3.4- dinámico

*dinámico* es un GNSA que mezcla 2 instancias distintas de las ecuaciones 16, 17 y 18 de [2] Gonzalez y otros para generar un entero sin signo de 32bits.

La ecuaciones básicas de *dinámico* son:

*Entradas:*

$x_n, y_n, z_n$

*Recurrencias:*

$$x_{n+1} = [(a_1+b_1y_n + c_1z_n)x_n + d_1y_n + e_1z_n] \pmod 1 \quad (16)$$

$$y_{n+1} = [(a_2+b_2z_n + c_2x_n)y_n + d_2z_n + e_2x_n] \pmod 1 \quad (17)$$

$$z_{n+1} = [(a_3+b_3x_n + c_3y_n)z_n + d_3x_n + e_3y_n] \pmod 1 \quad (18)$$

$$w = (x_{n+1} + y_{n+1} + z_{n+1}) \pmod 1$$

*Salidas:*

$w$

*dinámico1* y *dinámico2* tienen distintos valores en las constantes $a_1,a_2,a_3,b_1,b_2,b_3, c_1,c_2,c_3, d_1,d_2,d_3,e_1,e_2,e_3$ respectivamente. El valor resultante en *dinámico* es la suma de los 2 enteros sin signo, escalados a 32bits de los números doble precisión *w1* y *w2* calculados por *dinámico1* y *dinámico2*.

Una prueba de *dinámico* con *Tuftest* arroja estadísticos aceptables.

El código en C de *dinámico* y los resultados aparecen en el apéndice 2.

## 3-. GNSA en Criptografía

Como un manera de tratar de seguir las recomendaciones de diseño de los GNSA para ser usados en criptografía se ha escogido:

1.- Emplear, en lo posible, funciones difíciles de invertir o no invertibles
2.- Inicializar el GNSA por separado y al azar
3.- Incluir un calentamiento, si es posible aleatorio, inicial
4.- Reinicializar el estado interno al azar
5.- Decimar al azar los números generados; o sea descartar algunos números
6.- Usar tablas de mezcla para almacenar valores calculados previamente

Dado que los GNSA no han sido sometidos a ataques, y que aquí sólo se está probando su naturaleza seudo aleatoria se ha preferido denominarlos como generadores criptográficamente resistentes – CR -. Aguardando, a una vez verificada su resistencia a ataques, para calificarlos como Criptográficamente Seguros.

### 3.1.- porfiadoUL_CR

Este es el primer GNSA_CR propuesto en este documento. Se apoya en *porfiado4* como elemento generador del azar; y para hacerlo criptográficamente resistente se han incorporado al diseño los siguientes elementos.

Se almacenan de manera simétrica dos valores arrojados por *porfiado4*, previa decimación con una cierta probabilidad, en una tabla mezcla de $8*8$ en un esfuerzo por evitar la consecutividad en la sucesión generada; reponiendo los valores extraídos de manera asimétrica. Además se toma como valor generado $z = (x+y) -(int)(x+y)$ o sea la parte decimal de la suma indicada.

Cada cierto tiempo, al azar de acuerdo a los valores generados, se reinicia *porfiado4*.

La inicialización se hace por separado con un calentamiento inicial, en función de dos parámetros y de los valores iniciales de $x_0, y_0, F_0$ y $G_0$.

La subrutina porfiadoUL_CR() implanta el algoritmo *porfiadoUL_CR* que es la versión en entero sin signo de 32bits, criptográficamente resistente de *porfiado4*.

La organización es:

```
porfiadoUL_CR
    |
    |  // inicialización
    |---| iniciar_porfiado_CR
    |   | // calentar porfiado_CR y llenar Tabla de mezcla
    |   |--- porfiado4
    |
    |  // generación
    |---| decimar_porfiado
        |  |  // descartar aleatoriamente los números generados, prob= 0.125
        |  |--- porfiado4
        |
        |---| generar_porfiadoUL_CR
        |   |  // extraer al azar Tabla(i,j) y Tabla(j,i), reemplazar valores,
        |   |     devolver (x+y) - (int)(x+y)
        |   |--- porfiado4
        |
        |     // reinicialización aleatoria de porfiado4
        |--- reiniciar_porfiado_CR
```

Se hicieron 100 pruebas aleatorias e independientes de *porfiadoUL_CR* con *Tuftest* usando inicializaciones al azar e independientes del generador, con valores iniciales en los rangos indicados:

$x_0 \in [0.1,0.9], y_0 \in [0.1,0.9], Fx_0 \in [100,200], Gy_0 \in [100,200].$

Los valores decimales se llevaron a enteros UL.

Para hacer mas manejables los resultados estos valores estadísticos se categorizaron:

-1 cuando se van del rango permitido; 1 a 5 con valores permitidos.
El rango entero total es {-1, [4,20]} privilegiando los valores centrales según la siguiente función *rango*:

*valor = rango(**p**)* es:

Si ***p*** es un estadístico de *Tuftest* entonces

Si $\boldsymbol{p} \leq 0.01$ o $\boldsymbol{p} \geq 0.99$ entonces *valor = -1*
en caso contrario

Si $\boldsymbol{p} \le valor = [10*\boldsymbol{p}] + 1$
caso contrario $valor = 10 - [10*\boldsymbol{p}]$

*[x]* es la parte entera de *x*.

Los resultados obtenidos y la distribución de frecuencias se pueden apreciar en la tabla siguiente.

| clases | frecuencias |
|---|---|
| -1 | 3 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 1 |
| 8 | 3 |
| 9 | 4 |
| 10 | 8 |
| 11 | 6 |
| 12 | 13 |
| 13 | 16 |
| 14 | 15 |
| 15 | 12 |
| 16 | 9 |
| 17 | 7 |
| 18 | 3 |
| 19 | 0 |
| 20 | 0 |

Como se puede observar sólo en 3 casos se obtuvo un resultado cuestionable. Su moda es 13; el valor medio es 12.81.

La tabla con el detalle de los resultados figura en el apéndice 3.

Su resistencia criptográfica se apoya en la descripción siguiente:

Si se intercepta *z*, un cierto valor generado por *porfiadoUL_CR*, éste es el resultante de la expresión *(x+y) – [(x+y)]*, o sea la fracción decimal de esa suma con dos términos desconocidos *x* y *y*. Estos últimos valores residían en la tabla de mezcla, y en ella estaban ubicados de manera simétrica, *Tabla(i,j)* y *Tabla(j,i)*. Sin embargo el reemplazo se hace de manera asimétrica con lo que uno de ellos es sustituido mientras que el otro no. Por lo tanto al extraer dos elementos de la Tabla estos pertenecen a generaciones distintas de *porfiado4*. Mas aun se desconocen los valores de sus respectivas generaciones, en relación a la

sucesión arrojada por *porfiado4*, ya que estos permanecen en la tabla de acuerdo a un valor probabilístico.

Si además sabemos que antes de su generación son descartados o no de acuerdo al proceso de decimación, también probabilístico, todo esto hace que uno de los posibles sistemas de ecuaciones sea:

$$z = (x+y) - [(x+y)]$$
$$x = (100a) - [100a]$$
$$a = A/(x_{n+p+d} + Gy_{n+p+d})$$
$$y = (100b) - [100b]$$
$$b = B/(y_{m+q+c} + Fx_{m+q+c})$$

donde se conoce $z$ y se desconocen las demás cantidades.

Incluso $A$ y $B$ pueden ser distintos ya que se desconoce si se realizó en el ínterin alguna reinicialización, en ese caso estarán intercambiados. Los valores de los índices *n, p, d* y *m. q, c* son valores aleatorios dados por las distribución geométrica con $p = 1/64$ en el caso de la permanencia, y $p = 0.125$ cuando se trata de decimación. Incluso se debe contemplar que en la diagonal de la tabla, con probabilidad *1/8*, se realiza otra decimación al insertar un segundo valor ya que desaparece el valor colocado primero, *Fx*. Los valores de probabilidad de permanencia se pueden usar para plantear las ecuaciones más probables; mas esto es sólo una manera racional de plantear un ataque al generador, y por supuesto no garantiza el éxito.

En síntesis que el sistema anterior contiene *8* incógnitas, impidiendo su resolución sin mas datos.

## 3.2.- Kiss_me

*Kiss_me* es un GNSA_CR que se compone de dos instancias distintas de *Kiss99()*, con un calentamiento variable en ambos *Kiss99_o* y *Kiss99_b*, se usa una tabla de mezcla para almacenar los valores enteros generados internamente. Los dos números extraídos de la tabla son sumados módulo $2^{32}$. Adicionalmente se incluye una reinicialización variable.

La organización funcional es:

```
Kiss_me
        |   // inicialización
        |---| init_Kiss99_dual
        |   | // inicializar, calentar Kiss_me y llenar Tabla de mezcla
        |   |--- init_Kiss99_o
        |   |--- init Kiss99_b
        |   |--- Kiss99_o
        |   |--- Kiss99_b
        |
        |   // generar valores
        |---| Kiss99_dual
        |   |   // extraer al azar Tabla(i,j) y Tabla(j,i), reemplazar valores,
        |   |   // y devolver la suma módulo 2^32 como valor
        |   |--- Kiss99_o
        |   |--- Kiss99_b
        |
        |    // reinicilizacion aleatoria de Kiss_me
        |--- reiniciar_Kiss99_me
```

*Kiss99* es la nueva versión C en doble precisión del algoritmo *Kiss* de [11] Marsaglia, en Random Numbers for C: End, at last?

*Kiss99_o* y *Kiss99_b* son dos instancias distintas de *Kiss99* con algunos coeficientes cambiados en *SHR3* y otro congruencial lineal respecto a *Kiss99_o*.

En particular se tomaron las versiones siguientes de *Kiss99*. Estas en versión macro de C.

*Kiss99_o* es:

```
Kiss99_o = (CONG xor MWC) + SHR3 donde: // versión original

CONG: x == 69069*x+ 1234567 (mod 2**32)

MWC: zw = ((z<<16)+ w)
        z = 36969*(z&65535)+(z>>16)
```

w = 18000*(w&65535)+(w>>16)

SHR3:(jsr^=(jsr<<17) + 123, jsr^=(jsr>>13) + 456, jsr^=(jsr<<5) + 789)

y

*Kiss_b* es:

Kiss99_b = (CONG xor MWC) + SHR3 donde: // *versión nueva*

CONG: x == 1566083941*x+ 1234567 (mod 2**32) // otro GCL

MWC: zw = ((z<<16)+ w)
z = 36969*(z&65535)+(z>>16)
w = 18000*(w&65535)+(w>>16)

SHR3:(jsr^=(jsr<<16) + 123, jsr^=(jsr>>7) + 456, jsr^=(jsr<<11) + 789)

Se realizaron 98 pruebas independientes, con inicializaciones al azar, con *Tuftes*t. Los resultados estadísticos aparecen el apéndice 3.

Los resultados son desalentadores, la moda es –1 con frecuencia 39 lo que indica que casi el 38% de los casos ensayados tienen estadísticos que no aprueban la prueba *Tuftest*.

La media es 7.97.

Sin lugar a dudas este generador *Kiss_me* no cumple con las condiciones de azar, uniformidad e independencia que se requieren para su uso criptográfico.

La distribución de frecuencias es

| clases | frecuencias |
|---|---|
| -1 | 39 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 1 |
| 9 | 1 |
| 10 | 2 |
| 11 | 8 |
| 12 | 5 |
| 13 | 10 |
| 14 | 9 |
| 15 | 5 |
| 16 | 8 |
| 17 | 5 |
| 18 | 3 |
| 19 | 2 |
| 20 | 0 |

### 3.3. girar_me

*girar_me* es un GNSA_CR que se compone de 2 instancias distintas de *girar_dec()*, con un calentamiento variable y decimación en ambos *girar_dec1*y *girar_dec2*. Se usa una tabla de mezcla para almacenar el valor doble precisión generado mezclando bits de uno y otro. Dos números enteros son usados para indexar la posición de tabla de la cual se extrae el valor devuelto. La conversión a entero sin signo se hace después de extraer el valor doble de la tabla. Adicionalmente se incluye una reinicialización variable.

La organización es:

```
girar_me
        |   // inicialización
        |---| init_girar_me
        |   | // calentar girar_me y llenar Tabla de mezcla
        |   |--- init_girar_dec1
        |   |--- init girar_dec2
        |   |--- girar_dec1
        |   |--- girar_dec2
        |   |--- calcular_doble
        |
        |   // generar valores
        |---| girar_me
        |   |  // extraer al azar Tabla(i,j) y Tabla(j,i), reemplazar valores,
        |   |     y devolver valor doble
        |   |--- girar_dec1
        |   |--- girar_dec2
        |   |--- calcular_doble
        |
        |    // reinicialización aleatoria de Kiss_me
        |--- reiniciar_girar_me
```

*girar_dec1* y *girar_dec2* contienen el generador primario *girar*, y su decimación *girar_dec*:

*girar:*

```
entradas:
    a1,a2,a3,a4
calcular
      a1 = ~(a1^(a1<<5) + a2 + 1234567);
     a2 = a2^(a2>>13) + a1 + 76543;
```

```
        a3 = a3^(a3<<6)+ a4 + 314151;
        a4 = ~(a4^(a4>>7)+ a3 + 349765);

        z = (a1^a3) + (a2^a4);
  salida:
      z
```

donde ^ es el operador *xor* de C.

Se realizaron 100 pruebas independientes al azar con *Tuftest*. Los resultados aparecen el apéndice 3.

Los resultados son desalentadores, la moda es –1 con frecuencia 32 lo que indica que casi el 32% de los casos ensayados tienen estadísticos que no aprueban la prueba *Tuftest*. La media es 8.39.

Sin lugar a dudas este generador *girar_me* no cumple con las condiciones de azar, uniformidad e independencia que se requieren.

La distribución de frecuencias es:

| clases | frecuencias |
|---|---|
| -1 | 32 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 1 |
| 8 | 3 |
| 9 | 3 |
| 10 | 7 |
| 11 | 8 |
| 12 | 11 |
| 13 | 6 |
| 14 | 7 |
| 15 | 9 |
| 16 | 6 |
| 17 | 3 |
| 18 | 0 |
| 19 | 2 |
| 20 | 0 |

### 3.4.- dinámico_CR

*dinámico_CR* es un GNSA_CR que se compone de 2 instancias distintas de *dinámico()*, cada uno con un calentamiento variable.

*dinámico* es la versión en entero sin signo de 32 bits de las ecuaciones 16, 17 y 18, en doble precisión que transforman un sistema dinámico en uno simétrico.

La ecuaciones básicas de dinámico son:

$$x_{n+1} = [(a_1+b_1y_n + c_1z_n)x_n + d_1y_n + e_1z_n] \pmod 1 \quad (16)$$

$$y_{n+1} = [(a_2+b_2z_n + c_2x_n)y_n + d_2z_n + e_2x_n] \pmod 1 \quad (17)$$

$$z_{n+1} = [(a_3+b_3x_n + c_3y_n)z_n + d_3x_n + e_3y_n] \pmod 1 \quad (18)$$

tomadas de [2] González y otros.

Los valores $a_1,a_2,a_3,b_1,b_2,b_3,c_1,c_2,c_3,d_1,d_2,d_3,e_1,e_2,$ y $e_3$ son constantes reales mayores que 1.

$x_0$, $y_0$, y $z_0$ en *dinámico_g1*, y $xx_0$, $yy_0$, y $zz_0$ en *dinámico_g2* son los valores iniciales del algoritmo *dinámico_CR*.

En base a estos dos valores generados por *dinámico_g1* y *dinámico_g2* se construye como la suma el valor generado.

Se usa una tabla de mezcla para almacenar el valor entero generado. Adicionalmente se incluye una reinicialización variable.

Las organización de dinámico_CR es:

```
dinámico_CR
      |
      |   // inicialización
      |---| iniciar_dinámico_CR
      |   | // calentar dinámico_g1 y dinámico_g2 y llenar Tabla de mezcla
      |   |--- dinámico_g1
      |   |--- dinámico_g2
      |   |--- dinámico
      |
      |   // generar valores
      |---| generar_dinámico_CR
      |   |  // extraer al azar Tabla(i), reemplazar valor
      |   |--- dinámico_g1
      |   |--- dinámico_g2
      |   |--- dinámico
      |       |
      |       |--- dinámico1
      |       |--- dinámico_g1
      |       |--- dinámico2
      |       |--- dinámico_g2
      |
      |   // reinicialización aleatoria de dinámico_CR
      |--- reiniciar_dinámico_CR
```

El programa en C y los valores numéricos obtenidos aparecen en el apéndice 3.

Los resultados obtenidos luego de efectuar 100 pruebas al azar, independientes con *Tuftest* resultan insuficientes. Casi el 29% de los casos muestreados presentan estadísticos inaceptables, lo cual también lo inhabilita como un GNSA_CR.

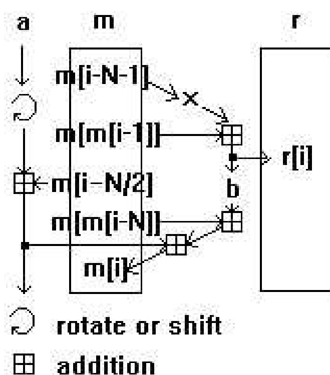
La distribución de frecuencias aparece a continuación.

| clases | frecuencias |
| --- | --- |
| -1 | 19 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 0 |
| 8 | 1 |
| 9 | 2 |
| 10 | 3 |
| 11 | 2 |
| 12 | 9 |
| 13 | 5 |
| 14 | 6 |
| 15 | 6 |
| 16 | 5 |
| 17 | 5 |
| 18 | 2 |
| 19 | 0 |
| 20 | 0 |

## 3.5.- isaac_CR

ISAAC es la abreviatura de Indirection, Shift, Accumulate, Add, and Count y genera un número aleatorio de 32-bit. Y está referido como un generador criptográficamente seguro por el autor en [3] R. Jenkins.

La figura siguiente tomada de [3] muestra un esquema de los cálculos en *isaac*.




La versión legible de *isaac* (*readable.c*), de las varias existentes, calcula un vector randrsl[0..255] de números enteros UL generados en cada invocación. Inicialmente se llama a *randinit*, rutina de inicialización, con una semilla fija en ella.

Dado que *Tuftest* requiere una rutina de generación de números aleatorios que en cada invocación devuelva un número de 32bits, se incorporaron dos rutinas adicionales *saca_uno* y *isaac_CR* para poder comprobar las propiedades aleatorias de *isaac* con *Tuftest*.

En *isaac_CR* se incorporaron las rutinas *saca_uno*, que como su nombre lo indica sirve para extraer un número del vector randrsl[0..255], en orden secuencial 1, 2, ..., 255, 0 hasta agotar el vector y luego invoca a *isaac* para rellenar con nuevos valores el vector randrsl.

Adicionalmente tiene una reinicialización por medio de los valores iniciales de las variables *a, b, c , d, e, f, g, h* de *mix(a,b,c,d,e,f,g,h)*, rutina usada en *randinit*.

*randinit* es la rutina de Jenkins para inicializar *isaac*, la cual es fija en su versión original.

La misma *isaac_CR* se inicializa cada vez que es invocada, y utiliza *saca_uno* para devolver un entero de 32 bits. La inicialización se hace vía *randinit* colocando valores externos para *mix*, y luego llamando a *isaac* para rellenar el vector randrsl[0..255].

Finalmente la organización de isaac_CR queda:

```
isaac_CR
    |// inicializar
    |--- randinit
    |       |
    |       |--- mix
    |       |--- isaac
    |
    |  // generar
    |--- saca_uno
            |
            |--- isaac
```

El programa en C y los valores numéricos obtenidos aparecen en el apéndice 3.

Los resultados obtenidos luego de efectuar 65 pruebas al azar, independientes con *Tuftes*t resultan insuficientes. Casi el 35% de los casos muestreados presentan estadísticos inaceptables. Es claro que *isaac_CR* no es una elección adecuada como GNSA_CR.

La distribución de frecuencias es:

| clases | **frecuencias** |
|---|---|
| -1 | **28** |
| 4 | **0** |
| 5 | **0** |
| 6 | **0** |
| 7 | **1** |
| 8 | **0** |
| 9 | **1** |
| 10 | **4** |
| 11 | **4** |
| 12 | **10** |
| 13 | **11** |
| 14 | **6** |
| 15 | **8** |
| 16 | **4** |
| 17 | **2** |
| 18 | **1** |
| 19 | **0** |
| 20 | **0** |

### 3.6.- reticulado_CR

Este GNSA implanta reticulado_CR como una versión de *reticulado* que usa dos instancias de *reticulado19* para construir el valor generado. r*eticulado19* es la versión en entero sin signo de 32 bits de la ecuación 19, en doble precisión de un reticulado acoplado..

La ecuación 19 es:

$$x_{n+1}(i) = [(a+bx_n(i-1)+cx_n(i+1))x_n(i) + dx_n(i-1) +fx_n(i+1) +0.1] \ (mod\ 1)$$

tomadas de [2] Gonzalez, Moreno y Guerrero.

Los valores *a, b, c, d, f* son constantes reales mayores que 1.

$x_0(-2), x_0(-1),x_0(0), x_0(1), x_0(2)$ son los valores iniciales del método.

*izq* (o sea inicialmente $x_0(-2)$) y *der* ( la primera vez $x_0(2)$) son dos GNSA congruenciales lineales módulo 1, que se necesitan para calcular los valores en los bordes y que no están especificados en el método original.

*izq = 2.3646323319990018e-1 * izq + 0.287445029243826e-4*
*izq = izq - (int)izq*

*der = 1.6081379726529121e-1 *der + 0.445029243987863e-4*
*der = der - (int)der*

El valor decimal z devuelto como generado es la suma de los términos centrales

$$z =(x_n(-1) + x_n(0) + x_n(1)) \ mod1$$

ya que como se vió antes el término original $x_n(0)$ no pasa *Tuftest.*

Se toman 2 instancias distintas *reticulado1* y *reticulado2*, que difieren en los valores de las constantes y en las condiciones de borde izquierda y derecha En base a estos dos valores generados por cada instancia, se construyen 2 enteros sin signo, luego la suma de estos, módulo 2**32 proporciona el valor entero UL generado por reticulado_CR.

Se emplea una tabla de mezcla para almacenar los valores enteros, desde donde se extrae al azar uno de ellos. Se reemplaza el valor extraído con un nuevo valor calculado.

También contiene una rutina de reinicialización que, al azar, intercambia los valores de los estados de las instancias reticulado1 y reticulado2.

Los resultados de las pruebas con *Tuftest* resultan una vez mas decepcionantes.

Casi el 40%(39.37) de los casos ensayados dan resultados inadecuados con el valor del estadístico -1. Esta condición experimental sugiere no utilizar este generador en Criptografía.

La distribución de frecuencias aparece a continuación.

Los respectivos valores numéricos se pueden consultar en el apéndice 3.

| clases | frecuencias |
|---|---|
| -1 | 50 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 1 |
| 8 | 3 |
| 9 | 3 |
| 10 | 6 |
| 11 | 12 |
| 12 | 7 |
| 13 | 9 |
| 14 | 12 |
| 15 | 11 |
| 16 | 3 |
| 17 | 4 |
| 18 | 4 |
| 19 | 2 |
| 20 | 0 |

### 3.7.- FF_CR

Este GNSA implanta *FF_CR* como una versión criptográficamente resistente de *FF*.

*FF* es un generador derivado de *porfiado4*, al igual que en éste se realizan las operaciones en doble precisión. Su núcleo de cálculo se apoya en una transformación no lineal. Primero se rotan los enteros derivados de los valores reales de entrada; y a estos se los lleva a números doble precisión. Tiene incluida una tabla de mezcla desde la que se extraen al azar los números doble precisión generados; y además una reinicialización al azar de acuerdo a la secuencia generada.

La transformación no lineal base es la recurrencia:

$$x_{n+1} = AA/(x_n + (1.0/(Fy_n+1.0) + 1.75432198))$$
$$y_{n+1} = AA/(y_n + (1.0/(Fx_n+1.0) + 3.14159265))$$

donde $AA = 151.4751$ y $BB = 173.4789$ son constantes reales en el intervalo $[100,200]$.

Los valores iniciales son valores reales $x_0=1.54321$, $y_0=2.56789$, $Fx_0= 5.0$ y $Fy_0=5.5$ *en el* intervalo $[1,20]$.

Su esquema de la recurrencia en *FF* es el siguiente:

Entradas:
*x, y, Fx, Fy*

1.- Tomar las mantisas de *x* y *y* en doble precisión
2.- Llevar esas mantisas a enteros grandes sin signo *n1* y *n2* (32bits)
3.- Hacer una rotación circular fija (11 y 15) de los enteros *n1* y *n2*
4.- Llevar esos enteros rotados a números reales doble precisión
5.- Calcular los nuevos *x* y *y* con la transformación no lineal
6.- Calcular *Fx* y *Fy* descartando las 3 primeras posiciones decimales de *x* y *y*
7.- Devolver *Fx*

Salidas:
*x, y, Fx, Fy*

*Output*
*Fx*

*FF_CR* es la versión en enteros grandes criptográficamente segura de *FF*.

El esquema de cálculo de *FF_CR* es;

```
FF_CR
   |  // inicializar
   |---init_FF_CR   // inicializa FF_CR
   |        |
   |        |--- FF     // calcula nuevo valor en doble precisión
   |
   |---FF_UL_CR   // calcula valor en entero grande
   |        |
   |        |--- FF     // calcula nuevo valor en doble precisión
   |
   |  // generar
   |---FF_UL_CR          // calcula valor en entero grande
   |        |
   |        |--- FF            // calcula nuevo valor en doble precisión
   |
   |--- reiniciar_FF_CR   // reinicializa estado del generador
```

Los resultados de las pruebas con *Tuftest* aparecen a continuación.  Este GNSA_CR pasa las pruebas y en consecuencia es aceptable su uso en Criptografía.

La distribución de frecuencias aparece a continuación. Los respectivos valores numéricos se pueden consultar en el apéndice 3.

| clases | frecuencias |
|---|---|
| -1 | 4 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |
| 7 | 3 |
| 8 | 3 |
| 9 | 0 |
| 10 | 12 |
| 11 | 8 |
| 12 | 15 |
| 13 | 9 |
| 14 | 20 |
| 15 | 16 |
| 16 | 4 |
| 17 | 8 |
| 18 | 3 |
| 19 | 1 |
| 20 | 1 |

La fortaleza criptográfica de *FF_CR* se apoya en un razonamiento similar al realizado en *porfiadoUL_CR*. Todo estriba en la dificultad en calcular la inversa del cálculo asociado a un cierto número generado.

Es decir si un número o una secuencia de números consecutivos, son capturados eventualmente estos no revelan gran información acerca del estado o estados previos de *FF_CR* que los produjo.

En primer lugar la consecutividad en la observación/captura no es indicativa de consecutividad en la generación. El tiempo de permanencia en la tabla de mezcla es una variable aleatoria, y si bien se puede utilizar su conocimiento para intentar revisar los estados de acuerdo a los valores de probabilidad, pero esto se hará sin ninguna certeza.

El sistema al cual se arriba será del tipo, conociendo el entero ***z*** :

$$n1 \leftarrow (2^{32}-1)mantisa(x_{n+p})$$
$$n2 \leftarrow (2^{32}-1)mantisa(y_{n+p})$$
$$n1 \leftarrow (n1>>11) \oplus (n1<<21)$$
$$n2 \leftarrow (n2>>15) \oplus (n2<<17)$$
$$x \leftarrow n1/2^{32}$$

$y \leftarrow n2/2^{32}$

$x \leftarrow A/(x + 1.0/(Fy + 1.0) + 1.75432198)$
$y \leftarrow B/(y + 1.0/(Fx + 1.0) + 3.14159265)$
$Fx \leftarrow mantisa(1000x)$
$Fy \leftarrow mantisa(1000y)$
$z \leftarrow (2^{32}-1)Fx$

donde $n + p$ indica alguna generación desconocida en la que se realizó el cálculo. Como existe reinicialización se desconoce si $x_{n+p}$ y $y_{n+p}$ provienen de sus mismas secuencias. Incertidumbre que se agrega a la indeterminación de la mantisa de *1000x*. Los *xor* empleados para calcular los enteros tampoco tienen inversa.

En síntesis que la reversión parece muy difícil de realizar.

## 4. Conclusiones

Se revisaron algunos generadores de números seudo aleatorios, y se construyeron otros. Cualquiera sea el caso se exploraron las propiedades aleatorias buscando que sean criptográficamente seguros.

Se ratifica que los GNSA congruenciales lineales son inadecuados para su empleo en Criptografía y digamos que también en Simulación cuando se requieren excelentes propiedades seudo aleatorias. Ciertos generadores, por caso *Kiss*, *reticulado, dinámico* e *ISAAC* que lucían promisorios para su empleo en Criptografía han resultado decepcionantes, al menos en las implantaciones realizadas.

Se comprobaron las propiedades estadísticas de siete (7) GNSA_CR construidos. Las pruebas empíricas realizadas permiten concluir que sólo dos (2) GNSA de todos los ensayados aparecen como criptográficamente resistentes. Vale destacar que una (1) prueba con *Tuftest*, de las aproximadamente 100 independientes realizadas a cada generador, involucra alrededor de $2^{31}$ invocaciones al generador que se está comprobando.

Los GNSA_CR encontrados son *porfiadoUL_CR* y *FF_CR*. Sus versiones simples, sin tomar las previsiones para uso criptográfico, *porfiado4* y *FF*, son adecuadas para su uso en Simulación y/o Montecarlo.

Ambos generadores emplean una clase de transformaciones reales no lineales y no aritmética entera como es lo usual.

Mención especial y cuidado se debe hacer de incluir un test/prueba del buen y correcto funcionamiento de las rutinas en C, bajo diferentes sistemas operativos y compiladores. La experiencia ha señalado que no todos los compiladores proporcionan resultados iguales y consistentes. Esto es importante cuando se piensa en su utilización cifrando en un ambiente operativo y descifrando en otro distinto.

**Referencias**

[1] E. Barker, y J. Kelsey (2005), “Recommendation for Random Number Generation Using Deterministic Random Bit Generators” NIST 800-90.

[2] J.A. Gonzalez , A.J. Moreno , y L.E. Guerrero (2005), “Non-invertible Transformations and Spatiotemporal Randomness”.

[3] R. J. Jr. Jenkins (1993-1996), http://burtleburtle.net/bob/rand/isaac.html# “IBAA ISAAC and RC4”.

[4] J. Kelsey, B. Schneier, D. Wagner, y C. Hall, (1999), “ Cryptanalytic Attacks on Pseudorandom Number Generators”.

[5] M. G. Kendall, y B. Babington-Smith (1938), "Randomness and Random Sampling Numbers," Journal of the Royal statistical Society, Series A, 101.

[6] D. Knuth, *The Art of Computer Programming*, (1998), ed. Addison Wesley, vol. 2: *Seminumerical Algorithms*, third edition,

[7] R. A. Lavieri (1984), “Exploración de nuevas ideas para generación de números pseudo-aleatorios” Trabajo de ascenso. Universidad Central de Venezuela, Facultad de Ciencias, Escuela de Computación. .

[8] L’ecuyer, P. and Simard, R. (2007). “TestU01: A C library for empirical testing of random number generators.” ACMTrans. Math. Softw. 33, 4, Article 22 (August 2007),

[9] G. Marsaglia (1985), “The Marsaglia Random Number CDROM, with The Diehard Battery of Tests of Randomness, “ . Access www.stat.fsu.edu/pub/diehard.

[10] G. Marsaglia “ KISS_f90.htm”

[11] G. Marsaglia (1999), “Random numbers for C: End, at last?”

[12] G . Marsaglia, and Wai Wan Tsang (2006), “Some difficult-to-pass tests of randomness”.

[13] R.A. Pastoriza (2007), “⊕P⊕ Un algoritmo aleatorio de cifrado simétrico por bloques”. RT-2007-11. Universidad Central de Venezuela. Facultad de Ciencias. Escuela de Computación. CIOMMA.

[14] Press, W. H., Teukolsky, S. A., Vetterling, W. T., y Flannery, B. P. (1996), *Numerical Recipes in Fortran 90*. Cambridge University Press. Vol.2..

[15] Rukhin, A., Soto,J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., San Vo. (2001), NIST 800-22 "A Statistical Test Suite For random and Pseudorandom Number Generators For Cryptographic Applications".

[16] Y. Sabbagh (1985), "Pruebas estadísticas para generadores de números aleatorios". Trabajo especial de Grado. Universidad Central de Venezuela, Facultad de Ciencias, Escuela de Computación.

## Apéndice 1 GNSA no tan buenos

En esta sección se incluyen los códigos y los resultados de las pruebas con Tuftest a aquellos GNSA que no han obtenido resultados aceptables con las sucesiones de números seudo aleatorios generados.

### 1.1.- Randu

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización $x_0 = 45813$.

```
/*
  Este archivo contiene un simple main() para llamar Randu_solo que es
  la versión en entero sin signo de Randu(). Randu() es una implantación
  del generador de xn+1 == 65539xn (mo0d 2**31),  que devuelve un valor
  aleatorio doble entre (0,1).

  programador R.A. Pastoriza        feb/2008
*/

#include<stdio.h>
//GNSA Randu
// valor inicial x
    long x=45813;
//////////////////////////////////////////////////////////////////////////////////////////
 double  Randu(void)
{
          double u;
//        printf(" en Randu entrando x== %u \n",x);
//                            2**31 = 2147483648
          x=((int)65539*(int)x)%(2147483648);        // el peor GNSA de todos los conocidos

          u = x*0.465661287307739e-9;
          return u;
}
//////////////////////////////////////////////////////////////////////////////////////////
unsigned long Randu_solo()
{
         double Randu();
         unsigned long w;
//   2**32 = 4294967296
         w = 4294967295*Randu();
         return(w);
}
```

```
////////////////////////////////////////////////////////////////////////////////////////
        int main(void)
 {
    unsigned long Randu_solo();
    unsigned long w;
    int i;
         printf("        prueba Randu \n");

         for(i=1;i<=10;i++)
         {
                  w = Randu_solo();
                 printf("iter =  %d numero en principal ==>   %u \n",i,w);
         }
         printf("\n           en 10 iteraciones  x10 es 2741535609  \n");
        printf("iter =  %d el numero en main es ==>   %u \n",i,w);
        return w;
 }
////////////////////////////////////////////////////////////////////////////////////////
```

```
Resultados Tuftest:
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Birthday spacings test: 4096 birthdays, 2^32 days in year
        Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed     0     0     0     0     0     0     0     0     0     0  5000
(O-E)^2/E  91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2604877.3
          Birthday Spacings: Sum(O-E)^2/E=609836.597, p= 1.000

Euclid's algorithm:
 p-value, steps to gcd:   1.000000
 p-value, dist. of gcd's: 1.000000

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 0.000 0.000 0.000 0.000 1.000 1.000 1.000 1.000
 Bits  8 to 15---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits 16 to 23---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits 24 to 31---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 KS test for the above 32 p values:  1.000
Press any key to continue

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

## 1.2.- ran1

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización $x_0 = 1234567$.

```
/*
  Este archivo contiene un simple main() para llamar ran que es
  la versión en entero sin signo de ran1(). ran1() es una implantación
  del generador de Lewis, Goodman, y Miller, con una tabla de mezcla,
  que devuelve un valor aleatorio real entre (0,1).

  programador R.A. Pastoriza       feb/2008
*/
#include <stdio.h>
// ran1 de [14] Press y otros, versión C de Numerical Recipes
//////////////////////////////////////////////////////////////////////////////////////////
#define IA 16807
#define IM 2147483647   // 2^31-1
#define AM (1.0/IM)
#define IQ 127773
#define IR 2836
#define NTAB 32
#define NDIV (1+(IM-1)/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0-EPS)

float ran1(long *idum)
{
        int j;
        long k;
        static long iy=0;
        static long iv[NTAB];
        float temp;

//                      printf(" numero *idum entrando en ran1 ==>   %u \n",*idum);
        if (*idum <= 0 || !iy) {
                if (-(*idum) < 1) *idum=1;
                else *idum = -(*idum);
                for (j=NTAB+7;j>=0;j--) {
                        k=(*idum)/IQ;
                        *idum=IA*(*idum-k*IQ)-IR*k;
                        if (*idum < 0) *idum += IM;
                        if (j < NTAB) iv[j] = *idum;
                }
```

```
                iy=iv[0];
        }
        k=(*idum)/IQ;
        *idum=IA*(*idum-k*IQ)-IR*k;
//                              printf(" numero *idum saliendo en ran1 ==>   %u \n",*idum);

if (*idum < 0) *idum += IM;
        j=iy/NDIV;
        iy=iv[j];
        iv[j] = *idum;
        if ((temp=AM*iy) > RNMX) return RNMX;
        else return temp;
}
#undef IA
#undef IM
#undef AM
#undef IQ
#undef IR
#undef NTAB
#undef NDIV
#undef EPS
#undef RNMX
/////////////////////////////////////////////////////////////////////////////////

     long z = 1234567;
unsigned long ran()
  {
//          float ran1();
     double yy;
          unsigned long u;

          yy = ran1(&z);
//              printf(" numero real en ran ==>   %10lf \n",yy);
          u = 4294967295*yy;
//              printf(" numero en ran ==>   %10u \n",u);
     return u;
  }
///////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////

        int main(void)
 {
       unsigned long ran();
```

```
        unsigned long w;
        int i;


          printf("         prueba ran1 \n");
          for(i=1;i<=10;i++)
          {
             w = ran();
                 printf("iter =  %d numero en principal ==>   %u \n",i,w);
          }

          printf("\n           en 10 iteraciones  x10 es 2882564608  \n");
          printf("iter =  %d el numero en main es ==>   %u \n",i-1,w);
          return w;
 }
```

////////////////////////////////////////////////////////////////////////////////////////////////////

Resultados Tuftest:

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Birthday spacings test: 4096 birthdays, 2^32 days in year
          Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed     0     0     0     0     0     0     0     0     0     0  5000
(O-E)^2/E  91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2604877.3
            Birthday Spacings: Sum(O-E)^2/E=609836.597, p= 1.000

Euclid's algorithm:
 p-value, steps to gcd:   1.000000
 p-value, dist. of gcd's: 1.000000

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 0.947 0.480 0.492 0.908 0.669 0.493 0.725 0.442
 Bits  8 to 15---> 0.191 0.096 0.958 0.594 0.334 0.329 0.418 0.401
 Bits 16 to 23---> 0.114 0.824 0.520 0.928 0.481 0.193 0.311 0.725
 Bits 24 to 31---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 KS test for the above 32 p values:  1.000
Press any key to continue
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

## 1.3.- cong1

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización $x_0 = 38011$.

```
/* Este archivo contiene un simple main() para llamar cong1 que es
   la versión en entero sin signo de congruencial lineal

   x == (69069*x + 1234567) mod2**32.

   programador R.A. Pastoriza       sep/2007
*/

#include <stdio.h>

typedef unsigned long      i32; /* entero sin signo de 32 bits */
typedef unsigned __int64   i64; /* entero sin signo de de 64 bits  */


#define MOD 4294967296i64
//  valor inicial de la semilla x
    unsigned long x = 38011;

// congruencial
          unsigned long cong1(unsigned long b)
          {              /* congruencial lineal x == (69069*x + 1234567) mod2**32  */
//             printf(" x entrando cong1  ==== %u\n",b);
      b = (i32)(((i32)(69069i64*(i64)x)%MOD) + 1234567i32)%MOD;
//             printf(" b saliendo cong1  ==== %u\n",b);
            return (b);
          }

////////////////////////////////////////////////////////////////////////////
//  programa de prueba
            unsigned long main(void)
{
      int i;

            printf("         prueba cong1 \n");
            for(i=1;i<11;i++)
            {
                     x = cong1(x);
               printf("iter i-> %d numero en main ==> %u \n\n",i, x);
            }
```

```
        printf("\n          en 10 iteraciones  x10 es 4152914525  \n");
        printf("iter =  %d el numero en main es ==>   %u \n",i-1,x);
    return (x);
}
```

```
////////////////////////////////////////////////////////////////////////////////
Resultados Tuftest:
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Birthday spacings test: 4096 birthdays, 2^32 days in year
        Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed     0     0     0     0     0     0     0     0     0     0  5000
(O-E)^2/E  91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2604877.3
          Birthday Spacings: Sum(O-E)^2/E=609836.597, p= 1.000

Euclid's algorithm:
 p-value, steps to gcd:   1.000000
 p-value, dist. of gcd's: 1.000000

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits  8 to 15---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits 16 to 23---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits 24 to 31---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 KS test for the above 32 p values:  1.000
Press any key to continue
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

**1.4.- cong2**

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización $x_0$ = 314151611.

```
/* Este archivo contiene un simple main() para llamar cong2 que es
   la versión en entero sin signo del congruencial lineal
   x == (1099087573*x + 71828182845) mod2**32.

   programador R.A. Pastoriza       sep/2007
*/

#include <stdio.h>

typedef unsigned long      i32; /* entero sin signo de 32 bits */
typedef unsigned __int64   i64; /* entero sin signo de de 64 bits  */


#define MOD 4294967296i64
//  valor inicial de la semilla x
    unsigned long x = 314151611;

// congruencial
        unsigned long cong2(unsigned long b)
        {    /* congruencial lineal x == (1099087573*x + 71828182845) mod2**32  */
             printf(" x entrando cong2  ==== %u\n",b);
          b = (i32)(((i32)(1099087573i64*(i64)x)%MOD) + 71828182845i32)%MOD;
             printf(" b saliendo cong2  ==== %u\n",b);
          return (b);
        }

        //  programa de prueba
//////////////////////////////////////////////////////////////////////////////

          unsigned long main(void)
{
     int i;

          printf("          prueba cong2 \n");
          for(i=1;i<=10;i++)
          {
                    x = cong2(x);
              printf("iter i-> %d numero en main ==> %u \n\n",i, x);
          }
```

```
        printf("\n         en 10 iteraciones  x10 es 4279496537   \n");
        printf("iter =  %d el numero en main es ==>   %u \n",i-1,x);
    return (x);
}
```

//////////////////////////////////////////////////////////////////////////////

Resultados Tuftest:

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Birthday spacings test: 4096 birthdays, 2^32 days in year
         Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed     0     0     0     0     0     0     0     0     0     0  5000
(O-E)^2/E  91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2604877.3
           Birthday Spacings: Sum(O-E)^2/E=609836.597, p= 1.000

Euclid's algorithm:
 p-value, steps to gcd:   1.000000
 p-value, dist. of gcd's: 1.000000

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits  8 to 15---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits 16 to 23---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits 24 to 31---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 KS test for the above 32 p values:  1.000
Press any key to continue
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

## 1.5.- cong3

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización $x_0$ = 314151611.

```
/* Este archivo contiene un simple main() para llamar cong3 que es
   la versión en entero sin signo del congruencial lineal
   x == (1664525*x + 132721788) mod2**32

   programador R.A. Pastoriza       sep/2007
*/

#include <stdio.h>

typedef unsigned long      i32; /* entero sin signo de 32 bits */
typedef unsigned __int64   i64; /* entero sin signo de de 64 bits  */


#define MOD 4294967296i64
//  valor inicial de la semilla x
    unsigned long x = 314151611;

// congruencial
          unsigned long cong3(unsigned long b)
          {    /* congruencial lineal x == (1664525*x + 132721788) mod2**32  */
              printf(" x entrando cong3  ==== %u\n",b);
     b = (i32)(((i32)(1664525i64*(i64)x)%MOD) + 132721788i32)%MOD;
              printf(" b saliendo cong3  ==== %u\n",b);
            return (b);
          }

/////////////////////////////////////////////////////////////////////////////

//  programa de prueba
            unsigned long main(void)
{
     int i;

            printf("          prueba cong3 \n");
            for(i=1;i<=10;i++)
            {
                      x = cong3(x);
               printf("iter i-> %d numero en main ==> %u \n\n",i, x);
            }
```

```
        printf("\n          en 10 iteraciones  x10 es 1029707323  \n");
        printf("iter =  %d el numero en main es ==>   %u \n",i-1,x);
    return (x);
}
```

```
///////////////////////////////////////////////////////////////////////////////
Resultados Tuftest
++++++++++++++++++++++++++++++++++++++++++++++++++++
Birthday spacings test: 4096 birthdays, 2^32 days in year
        Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed     0     0     0     0     0     0     0     0     0     0  5000
(O-E)^2/E  91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2604877.3
           Birthday Spacings: Sum(O-E)^2/E=609836.597, p= 1.000

Euclid's algorithm:
 p-value, steps to gcd:   1.000000
 p-value, dist. of gcd's: 1.000000

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits  8 to 15---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits 16 to 23---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits 24 to 31---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 KS test for the above 32 p values:  1.000
Press any key to continue
++++++++++++++++++++++++++++++++++++++++++++++++++++
```

**1.6.- cong4**

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización $x_0$ = 2122560859.

```
/* Este archivo contiene un simple main() para llamar cong4 que es
   la versión en entero sin signo del congruencial lineal
   x == (1566083941*x + 1234567) mod2**32

   programador R.A. Pastoriza       sep/2007
*/

#include <stdio.h>

typedef unsigned long      i32; /* entero sin signo de 32 bits */
typedef unsigned __int64   i64; /* entero sin signo de de 64 bits  */


#define MOD 4294967296i64
//  valor inicial de la semilla x
   unsigned long x = 2122560859;

// congruencial
          unsigned long cong4(unsigned long b)
          {    /* congruencial lineal x == (1566083941*x + 1234567) mod2**32  */
              printf(" x entrando cong4  ==== %u\n",b);
     b = (i32)(((i32)(1566083941i64*(i64)x)%MOD) + 1234567i32)%MOD;
              printf(" b saliendo cong4  ==== %u\n",b);
            return (b);
          }

/////////////////////////////////////////////////////////////////////////////

//  programa de prueba
            unsigned long main(void)
{
     int i;

            printf("          prueba cong4 \n");
            for(i=1;i<=10;i++)
            {
                      x = cong4(x);
               printf("iter i-> %d numero en main ==> %u \n\n",i, x);
            }
```

```
        printf("\n          en 10 iteraciones  x10 es 2261063445  \n");
        printf("iter =  %d el numero en main es ==>   %u \n",i-1,x);
    return (x);
}
```

```
/////////////////////////////////////////////////////////////////////////////
Resultados Tuftest
++++++++++++++++++++++++++++++++++++++++++++++++++
Birthday spacings test: 4096 birthdays, 2^32 days in year
        Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed     0     0     0     0     0     0     0     0     0     0  5000
(O-E)^2/E  91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2604877.3
          Birthday Spacings: Sum(O-E)^2/E=609836.597, p= 1.000

Euclid's algorithm:
 p-value, steps to gcd:   1.000000
 p-value, dist. of gcd's: 1.000000

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits  8 to 15---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits 16 to 23---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits 24 to 31---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 KS test for the above 32 p values:  1.000
Press any key to continue
++++++++++++++++++++++++++++++++++++++++++++++++++
```

**1.7.- SHR3**

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización $x_0$ = 123456789.

```
/* Este archivo contiene un simple main() para llamar SHR3 que es
   la versión en entero sin signo de 32 bits del corrimiento de
   registros de  [5] Marsaglia, incluido en Kiss99.
   SHR3: (jsr^=(jsr<<17), jsr^=(jsr>>13), jsr^=(jsr<<5))

            programador R.A. Pastoriza       sep/2007
*/

#include <stdio.h>
// valor inicial de la semilla x
        unsigned long x =123456789;
//////////////////////////////////////////////////////////////////////////////
       unsigned long SHR3(unsigned long y)
                    {
                     unsigned long z;

//                             printf("\n y entrando SHR3 ==> %u  \n", y);
                     z=      y<<17;
//              printf(" z== %u \n", z);
         y = y^z;
//              printf(" y1== %u \n", y);

                     y = y^(y>>13);
//                             printf(" y2== %u \n", y);

                     y = y^(y<<5);
//                             printf(" y3== %u \n", y);
                    return y;
                    }
//////////////////////////////////////////////////////////////////////////////
// programa de prueba

int main(void)

{
   int i;

   for(i=1;i<=10;i++)
```

```
        {
                x=SHR3(x) ;
                printf( " \n i== %u x==> %u\n",i,x);
        }
         printf("\n          en 10 iteraciones  x10 es 669254918  \n");
         printf("          i==> %d  x ==> %u\n",i-1,x);

   return(x);

}
////////////////////////////////////////////////////////////////////////////////
```

Resultados Tuftest

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++
Birthday spacings test: 4096 birthdays, 2^32 days in year
          Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed     0     0     0     0     0     0     0     0     0     0  5000
(O-E)^2/E  91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2604877.3
            Birthday Spacings: Sum(O-E)^2/E=609836.597, p= 1.000

Euclid's algorithm:
 p-value, steps to gcd:   1.000000
 p-value, dist. of gcd's: 1.000000

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits  8 to 15---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits 16 to 23---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 Bits 24 to 31---> 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
 KS test for the above 32 p values:  1.000
Press any key to continue
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

## 1.8.- reticulado 19

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización:

aa =2.0, bb=1.0, cc=1.0 , dd= 1.0,ff= 1.0
alfa=0.1, beta= 0.1, gama= 0.1, izq= 0.1, der= 0.1

```
/* Este archivo contiene un simple main() para implantar reticulado1
   que es la versión en entero sin signo de 32 bits de la ecuación 19
   de un reticulado acoplado.

   La ecuación 19 es:

   xn+1(i) = [(a+bxn(i-1)+cx(i+1))xn(i) + dxn(i-1) +fxn(i+1) +0.1] (mod 1)

            programador R.A. Pastoriza       sep/2007
*/
#include <stdio.h>
//   [2] Gonzalez, Moreno y Guerrero, Non-invertible Transformations and Spaciotemporal
Randomness
//   ecuacion 19
//   xn+1(i) = [(a+bxn(i-1)+cx(i+1))xn(i) + dxn(i-1) +fxn(i+1) +0.1] (mod 1)
//   se incluyen dos ecuaciones de borde no reportadas en el articulo.
//   izq = (3.123*izq + 0.77) (mod 1)
//   der =  (1.2731*der + 1.1317) (mod 1)

// valores iniciales
    double aa =2.0, bb=1.0, cc=1.0 , dd= 1.0,ff= 1.0;
    double alfa=0.1, beta= 0.1, gama= 0.1, izq= 0.1, der= 0.1;

//////////////////////////////////////////////////////////////////////////////
    double reticulado_g1(void)
           {
      izq = 3.123*izq + 0.77;         // X(-2)
      izq = izq - (int)izq;  // la condicion de borde izquierda

     der = 1.2731*der + 1.1317;   // X(2)
     der = der - (int)der;  // la condicion de borde derecha

     alfa = ((aa +bb*izq + cc*beta)*alfa + dd*izq + ff*beta + 0.1);
     alfa = alfa -(int)alfa;            // X(-1)

     beta = ((aa + bb*alfa + cc*gama)*beta + dd*alfa + ff*gama + 0.1);
```

```
 beta = beta -(int)beta;          // X(0)

  gama = ((aa + bb*beta + cc*der)*gama + dd*beta + ff*der + 0.1);
  gama = gama - (int)gama;     // X(1)

         return beta;
        }
//////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////
        unsigned long reticulado1(void)
        {
                unsigned long ww;

                double reticulado_g1();

                ww = 4294967295*reticulado_g1();

                return ww;
        }
//////////////////////////////////////////////////////////////////////////////

// programa de prueba

int main(void)

{
   int i;
        unsigned long x;

        for(i=1;i<=10;i++)
        {

                x= reticulado1() ;
                printf( " \n i== %u x==> %u\n",i,x);
        }
         printf("\n en 10 iteraciones  x10 es 3892991097  \n");

         printf("           i==> %d  x ==> %u\n",i-1,x);

   return(x);

}

////////////////////////////////////////////////////////////////////////////////////////
```

Resultados Tuftest

```
+++++++++++++++++++++++++++++++++++++++++++++
Birthday spacings test: 4096 birthdays, 2^32 days in year
        Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9  >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed   116   321   719  1005   938   791   502   342   158    71    37
(O-E)^2/E  6.5   5.6   0.3   0.8   1.5   0.1   0.7   6.6   0.6   0.4   0.3
          Birthday Spacings: Sum(O-E)^2/E= 23.373, p= 0.991

Euclid's algorithm:
 p-value, steps to gcd:   0.749884
 p-value, dist. of gcd's: 0.576543

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 0.600 0.309 0.297 0.992 0.592 0.249 0.961 0.627
 Bits  8 to 15---> 0.529 0.794 0.071 0.470 0.505 0.615 0.434 0.113
 Bits 16 to 23---> 0.775 0.874 0.011 0.704 0.529 0.505 0.115 0.475
 Bits 24 to 31---> 0.666 0.809 0.842 0.255 0.601 0.761 0.594 0.948
 KS test for the above 32 p values:  0.587
Press any key to continue
+++++++++++++++++++++++++++++++++++++++++++++++++++++
```

### 1.9.- Kiss99_o

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización:

x = 380116160, y = 123456789, z = 362436069, w = 521288629

```
/* Este archivo contiene un simple main() para llamar Kiss99 que es
   la versión en entero sin signo del generador compuesto
   según [6] Marsaglia, Random numbers for C: End, at last? 1999

   Kiss = (CONG xor MWC) + SHR3 donde:

   CONG: x == 69069*x+ 1234567 (mod 2**32)

   MWC: zw = ((z<<16)+ w)
            z = 36969*(z&65535)+(z>>16)
            w = 18000*(w&65535)+(w>>16)

   SHR3:(jsr^=(jsr<<17), jsr^=(jsr>>13), jsr^=(jsr<<5))

      programación:
            R.A. Pastoriza septiembre 2007
 */

/*
   The KISS generator, (Keep It Simple Stupid), is
   designed to combine the two multiply-with-carry
   generators in MWC with the 3-shift register SHR3 and
   the congruential generator CONG, using addition and
   exclusive-or. Period about 2^123.
   It is one of my favorite generators.

   The MWC generator concatenates two 16-bit multiply-
   with-carry generators, x(n)=36969x(n-1)+carry,
   y(n)=18000y(n-1)+carry mod 2^16, has period about
   2^60 and seems to pass all tests of randomness. A
   favorite stand-alone generator---faster than KISS,
   which contains it.


   SHR3 is a 3-shift-register generator with period
   2^32-1. It uses y(n)=y(n-1)(I+L^17)(I+R^13)(I+L^5),
   with the y's viewed as binary vectors, L the 32x32
```

```
   binary matrix that shifts a vector left 1, and R its
   transpose. SHR3 seems to pass all except those
   related to the binary rank test, since 32 successive
   values, as binary vectors, must be linearly
   independent, while 32 successive truly random 32-bit
   integers, viewed as binary vectors, will be linearly
   independent only about 29% of the time.

   CONG is a congruential generator with the widely used 69069
   multiplier: x(n)=69069x(n-1)+1234567. It has period
   2^32. The leading half of its 32 bits seem to pass
   tests, but bits in the last half are too regular.

*/

#include <stdio.h>
#define  MOD  4294967296
//                valores iniciales
       unsigned long x= 380116160, y=123456789, z=362436069, w=521288629 ;

                    unsigned long kiss99_o(unsigned long a, unsigned long b, unsigned long c,
unsigned long d)

                    {//  el nombre Kiss99_o responde a que se toman los valores originales de
los parametros en Cong, SHR3 y MWC
                    unsigned long zw;
                    unsigned long kiss;

//       printf(" \n ***  x y z w entrando kiss99_O ==> %u %u %u %u\n ",a,b,c,d);

/* GCL: x == 69069*x+ 1234567 (mod 2**32) */

       a = (69069*a+1234567)%MOD;

//                        printf(" *** a dentro==>  %u\n",a);
                    x = a;

/*SHR3: (jsr^=(jsr<<17), jsr^=(jsr>>13), jsr^=(jsr<<5)) */

                      b = b^(b<<17);
                      b = b^(b>>13);
                      b = b^(b<<5);
//                        printf(" *** y dentro==>  %u\n",b);
         y = b;
```

```
/* MWC: multiply with carry  */
       c=36969*(c&65535)+(c>>16);
//                                    printf(" *** c dentro==>  %u\n",c);
                  z= c;
       d=18000*(d&65535)+(d>>16);
//                                    printf(" *** d dentro==>  %u\n",d);
                  w = d;
       zw = ((z<<16)+ w);
//                        printf(" *** zw dentro==>  %u\n",zw);

/*  kiss = (CONG xor MWC) + SHR3          */
       kiss = (zw^x) + y;
//                printf(" *** kiss dentro==>  %u\n",kiss);
      return (kiss);
           }
//////////////////////////////////////////////////////////////////////////
//  programa de prueba

            void main(void)
            {
        int i;
                    unsigned long hh;
        unsigned long Kiss99_o();

        for(i=1;i<=10;i++)
                    {
                            hh = kiss99_o(x,y,z,w);
                            printf(" i==> %u x==> %u\n", i,hh);
                    }
           printf("\n          en 10 iteraciones  x10 es  874366052  \n");
           printf("iter =  %d el numero en main es ==>   %u \n",i-1,hh);

            }
//////////////////////////////////////////////////////////////////////////
```

Resultados Tuftest

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++
Birthday spacings test: 4096 birthdays, 2^32 days in year
         Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed    88   382   730   963   968   835   523   279   123    78    31
(O-E)^2/E   0.1   0.7   0.0   0.2   0.1   3.7   0.0   1.2   4.5   2.1   2.3
           Birthday Spacings: Sum(O-E)^2/E= 14.852, p= 0.862
```

Euclid's algorithm:
 p-value, steps to gcd:   0.409427
 p-value, dist. of gcd's: 0.788435

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 0.414 0.845 0.300 0.292 0.452 0.970 0.861 0.877
 Bits  8 to 15---> 0.090 0.565 0.988 0.214 0.901 0.514 0.512 0.352
 Bits 16 to 23---> 0.843 0.161 0.699 0.996 0.127 0.744 0.597 0.272
 Bits 24 to 31---> 0.702 0.827 0.784 0.852 0.440 0.478 0.659 1.000
 KS test for the above 32 p values:  0.994
Press any key to continue
++++++++++++++++++++++++++++++++++++++++++++++++++++++

## Apéndice 2 GNSA que pasan Tuftest

En esta sección se incluyen los códigos y los resultados de las pruebas con Tuftest a aquellos GNSA que han obtenido resultados aceptables con las sucesiones de números seudo aleatorios generados.

### 2.1.- porfiado

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización:

x = 151.4751, y = 173.4789

```
#include<stdio.h>

        double x=151.4751, y=173.4789;
         double Fx, Gy;
/////////////////////////////////////////////////////////////////////////////

        double porfiado4(void)
        {
                double A=  151.4751;
               double B=  173.4789;

                Fx = (100*x) - (int)(100*x);
                Gy = (100*y) - (int)(100*y);

                x = A/(x+Gy);
                y = B/(y+Fx);

                return Fx;
        }
/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////

unsigned long porfiado(void)
   {
      double porfiado4();
      double yy;
     unsigned long u;

            yy = porfiado4();
//                printf(" numero real en porfiado ==>   %10lf \n",yy);
```

```
             u = 4294967295*yy;
//                printf(" numero en porfiado ==>   %10u \n",u);
       return u;
   }
/////////////////////////////////////////////////////////////////////////

//  programa de prueba
           unsigned long main(void)
{
      int i;
      unsigned long z;

           printf("         prueba porfiado4 \n");

           for(i=1;i<=10;i++)
           {
                     z = porfiado();
           printf("iter i-> %d numero en main ==> %u \n\n",i, z);
           }

           printf("\n           en 10 iteraciones  x10 es 1811384038   \n");
           printf("iter =  %d el numero en main es ==>   %u \n",i-1,z);

      return (z);
}

/////////////////////////////////////////////////////////////////////////
```

Resultados Tuftest

```
++++++++++++++++++++++++++++++++++++++++++++++++++++
Birthday spacings test: 4096 birthdays, 2^32 days in year
          Table of Expected vs. Observed counts:
Duplicates  0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed   104   365   751   958   958   805   497   314   148    63    37
(O-E)^2/E   1.7   0.0   0.5   0.4   0.4   0.7   1.1   0.9   0.0   0.2   0.3
            Birthday Spacings: Sum(O-E)^2/E=  6.066, p= 0.190

Euclid's algorithm:
 p-value, steps to gcd:   0.722859
 p-value, dist. of gcd's: 0.047552

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 0.732 0.711 0.668 0.611 0.881 0.301 0.205 0.941
```

```
 Bits  8 to 15---> 0.152 0.234 0.814 0.796 0.626 0.792 0.464 0.628
 Bits 16 to 23---> 0.355 0.635 0.379 0.183 0.930 0.095 0.682 0.250
 Bits 24 to 31---> 0.629 0.988 0.504 0.091 0.215 0.621 0.757 0.412
 KS test for the above 32 p values:  0.440
Press any key to continue
++++++++++++++++++++++++++++++++++++++++++++++++++++
```

**2.2.- Kiss99_b**

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización:

x1=3811617,x2=1234567,x3=314151617,x4=5278639;

```
#include <stdio.h>
/*

  Este archivo contiene una subrutina Kiss99_b() para implantar un generador
  que es la version en entero sin signo de 32bits; Kiss99_b
  tiene los coeficientes cambiados respecto a KIss99_o.

  Kiss99 es la version C en doble precision del algoritmo Kiss de [6] Marsaglia, en
  Random Numbers for C: End, at last? From: George Marsaglia <geo@stat.fsu.edu>
  Date: Thu, 21 Jan 1999 03:08:52 GMT
  Organization: Florida State University

  El algoritmo Kiss99 sigue la descripción siguiente:

  The KISS generator, (Keep It Simple Stupid), is
  designed to combine the two multiply-with-carry
  generators in MWC with the 3-shift register SHR3 and
  the congruential generator CONG, using addition and
  exclusive-or. Period about 2^123.
  It is one of my favorite generators.

  The MWC generator concatenates two 16-bit multiply-
  with-carry generators, x(n)=36969x(n-1)+carry,
  y(n)=18000y(n-1)+carry mod 2^16, has period about
  2^60 and seems to pass all tests of randomness. A
  favorite stand-alone generator---faster than KISS,
  which contains it.


  SHR3 is a 3-shift-register generator with period
  2^32-1. It uses y(n)=y(n-1)(I+L^17)(I+R^13)(I+L^5),
  with the y's viewed as binary vectors, L the 32x32
  binary matrix that shifts a vector left 1, and R its
  transpose. SHR3 seems to pass all except those
  related to the binary rank test, since 32 successive
  values, as binary vectors, must be linearly
  independent, while 32 successive truly random 32-bit
```

```
  integers, viewed as binary vectors, will be linearly
  independent only about 29% of the time.

  CONG is a congruential generator with the widely used 69069
  multiplier: x(n)=69069x(n-1)+1234567. It has period
  2^32. The leading half of its 32 bits seem to pass
  tests, but bits in the last half are too regular.

  programador R.A. Pastoriza       oct/2007
*/

// valores inicales
                unsigned long x_b,jsr_b,z_b,w_b;
                unsigned long x1=3811617,x2=1234567,x3=314151617,x4=5278639;
                int vez =1;
////////////////////////////////////////////////////////////////////////////////

                int maximo(int a,int b)
        {// maximo de los 2 valores enteros de entrada
                int z;

                if(a<=b) z=b;
                else z=a;

                return z;
        }

////////////////////////////////////////////////////////////////////////////////

   unsigned long Kiss99_b(void)      // otros coeficientes en Kiss; en Random Numbers for
C: End, at last?
{
          unsigned long y_b;

      x_b = 1566083941*x_b + 1234567;            //  otro GCL

      jsr_b ^= (jsr_b<<16) + 123;    // el SHR3  (16,-7,11) de Pres y otros
      jsr_b ^= (jsr_b>>7)  + 456;    // mas términos independientes
      jsr_b ^= (jsr_b<<11) + 789;

      z_b=36969*(z_b&65535)+(z_b>>16) + 34567;    // los MWC originales mas
      w_b=18000*(w_b&65535)+(w_b>>16) + 76543; // términos independientes
     y_b= (z_b<<16)+w_b;
```

```
        return ((x_b^y_b)+jsr_b);                    // valor final
}

////////////////////////////////////////////////////////////////////////////

      void init_Kiss99_b(unsigned long a1,unsigned long a2,unsigned long a3,unsigned long
a4)
          {// inicializacion Kiss99_b con calentamiento
                      unsigned long Kiss99_b();
//                   int maximo();

                      int k=27;
                      int kk;
                      int i;

                      (x_b= a1, jsr_b=  a2, z_b= a3, w_b= a4);

                      kk = (x_b+jsr_b+z_b+w_b)&63;

                      kk = maximo(k,kk);

                      for(i=1;i<kk;i++)
                      {// calentamiento
                                Kiss99_b();
                      }
            }
////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
      unsigned long Kiss99_bb(void)
            {
               unsigned long h;

               if(vez==1)
                      {
                         init_Kiss99_b(x1,x2,x3,x4);
                                vez =2;
//                    printf(" vez es === %i \n",vez);
                                h= Kiss99_b();
                      }

                      else
                      {
//                              printf(" entro en vez=2 \n");
```

```
                h= Kiss99_b();
            }

            return h;
        }
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

//  programa de prueba
        unsigned long main(void)
{
    int i;
    unsigned long z;

        printf("        prueba Kiss99_bb \n");
        for(i=1;i<=10;i++)
        {
           z = Kiss99_bb();

           printf("iter i-> %d numero en main ==> %u \n\n",i, z);
        }

        printf("\n          en 10 iteraciones  x10 es 872929142  \n");
        printf("iter =  %d el numero en main es ==>   %u \n",i-1,z);

    return (z);
}

//////////////////////////////////////////////////////////////////////////////
```

Resultados Tuftest
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

```
Birthday spacings test: 4096 birthdays, 2^32 days in year
        Table of Expected vs. Observed counts:
Duplicates    0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed     95   345   712  1013   991   771   531   283   161    52    46
(O-E)^2/E   0.1   1.2   0.6   1.3   0.2   0.1   0.2   0.7   1.0   3.0   0.7
          Birthday Spacings: Sum(O-E)^2/E=  9.274, p= 0.494
```

Euclid's algorithm:

```
 p-value, steps to gcd:   0.847521
 p-value, dist. of gcd's: 0.101839

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 0.790 0.536 0.490 0.402 0.550 0.598 0.039 0.898
 Bits  8 to 15---> 0.980 0.370 0.240 0.251 0.605 0.855 0.172 0.189
 Bits 16 to 23---> 0.444 0.920 0.366 0.668 0.245 0.185 0.431 0.838
 Bits 24 to 31---> 0.748 0.492 0.727 0.325 0.454 0.507 0.992 0.811
 KS test for the above 32 p values:  0.412
Press any key to continue
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

## 2.3.- girar_dec

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización:

a=123459; b=3801161; c=91234577; d=61123458;

```
/*
  girar_dec contiene girar, el generador primario, y su decimación:

  entrada:
        a1,a2,a3,a4
  calcular
                 a1 = ~(a1^(a1<<5) + a2 + 1234567);
                 a2 = a2^(a2>>13) + a1 + 76543;

                 a3 = a3^(a3<<6)+ a4 + 314151;
                 a4 = ~(a4^(a4>>7)+ a3 + 349765);

                 z = (a1^a3) + (a2^a4);
  salida:
        z

  programador R.A. Pastoriza       nov/2007
 */
#include <stdio.h>

                 unsigned long a=123459;
                 unsigned long b=3801161;
                 unsigned long c=91234577;
                 unsigned long d=61123458;

unsigned long girar_dec(void)
         {// otra relacion de decimacion mas alta

                 int k=7;

        unsigned long z;
        unsigned long x[4];

                 int i,j,kk,veces;

                 (x[1]=a,x[2]=b,x[3]=c,x[4]=d);
```

```
                kk = 0;  // inicio decimacion
                for(i=1;i<=4;i++)
                {
                        j = x[i]&15;
                        kk = kk + j;
                }

                k = (k+kk)%16;
        veces =1;
                if(k>=12) veces = 2; //decimacion 12/16=0.75     4/16=0.25

//              printf(" veces = %i  \n",veces);
        for(i=1; i<=veces;i++)
                {
                a = ~(a^(a<<5) + b + 1234567);
        b = b^(b>>13) + a + 76543;

                c = c^(c<<6)+ d + 314151;
                d = ~(d^(d>>17)+ c + 349765);

                z = (a^c) + (b^d);
        }

        return (z);
           }

//////////////////////////////////////////////////////////////////////////////


//  programa de prueba
           unsigned long main(void)
{
     int i;
     unsigned long z;

           printf("          prueba girar_dec \n");
           for(i=1;i<=10;i++)
           {
                z = girar_dec();
              printf("iter i-> %d numero en main ==> %u \n\n",i, z);
           }

           printf("\n            en 10 iteraciones  x10 es 4032129441   \n");
           printf("iter =  %d el numero en main es ==>   %u \n",i-1,z);
```

```
    return (z);
}

////////////////////////////////////////////////////////////////////////////////
 Resultados Tuftest

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Birthday spacings test: 4096 birthdays, 2^32 days in year
         Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed    87   353   759   969   954   792   526   311   160    46    43
(O-E)^2/E   0.2   0.5   0.9   0.1   0.5   0.1   0.0   0.6   0.8   6.1   0.1
           Birthday Spacings: Sum(O-E)^2/E= 10.154, p= 0.573

Euclid's algorithm:
 p-value, steps to gcd:   0.468684
 p-value, dist. of gcd's: 0.119148

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 0.480 0.021 0.695 0.580 0.476 0.253 0.089 0.945
 Bits  8 to 15---> 0.416 0.849 0.305 0.111 0.332 0.189 0.605 0.051
 Bits 16 to 23---> 0.125 0.304 0.761 0.844 0.479 0.958 0.415 0.940
 Bits 24 to 31---> 0.069 0.594 0.455 0.514 0.624 0.654 0.849 0.026
 KS test for the above 32 p values:  0.172
Press any key to continue
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

## 2.4.- dinámico

A continuación aparece el código en C y los resultados de correr la prueba Tuftest a la sucesión generada con la inicialización:

```
// valores iniciales de dinámico1, las constantes originales de [1] Gonzaleza y otros.
    double a1 =1.3, b1=3.14159265358979323, c1=2.6 , d1= 1.5, e1= 1.1;
    double a2 =4.6, b2=2.1, c2=1.1 , d2= 2.2, e2= 7.1;
    double a3 =2.9, b3=5.4, c3=8.7 , d3= 4.5, e3= 1.9;

// valores iniciales de las variables de estado dinámico1
   static double xn= 0.1, yyn= 0.1, zn= 0.1;


// valores iniciales de dinámico2.
        double aa1 =2.11345, bb1=1.141592653, cc1=1.776666 , dd1= 2.5789123, ee1=
4.98712345;
         double aa2 =3.6987065, bb2=0.312435, cc2=2.19998765 , dd2= 2.2468, ee2=
5.1234567;
    double aa3 =5.909876503, bb3=3.455667788, cc3=1.789789789 , dd3= 3.5654321, ee3=
3.901234567;
// valores iniciales de las variables de estado dinámico2
            static double xxn= 0.1, yyyn= 0.1, zzn= 0.1;


#include <stdio.h>
//  dinámico de [2] Gonzalez y otros GNSA33

/* Este archivo contiene un simple main() para implantar dinámico
   que es la version en entero sin signo de 32 bits de las ecuaciones
   16, 17 y 18 que transforman un sistema dinámico en uno simetrico.

   La ecuaciones de dinámico son:

         xn+1 = [(a1+b1yn + c1zn)xn + d1yn + e1zn] (mod 1)    (16)

         yn+1 = [(a2+b2zn + c2xn)yn + d2zn + e2xn] (mod 1)    (17)

         zn+1 = [(a3+b3xn + c3yn)zn + d3xn + e3yn] (mod 1)    (18)

   tomadas de [2] Gonzalez, Moreno y Guerrero, Non-invertible Transformations
   and Spaciotemporal Randomness. 2005
```

```
 Se toman 2 instancias distintas dinámico1 y dinámico2. En base a estos
 valores se construye como la suma el valor generado.

  programador R.A. Pastoriza       sep/2007
*/
/////////////////////////////////////////////////////////////////////////////

// valores iniciales de dinámico1, las constantes originales de Gonzaleza y otros.
   double a1 =1.3, b1=3.14159265358979323, c1=2.6 , d1= 1.5, e1= 1.1;
   double a2 =4.6, b2=2.1, c2=1.1 , d2= 2.2, e2= 7.1;
   double a3 =2.9, b3=5.4, c3=8.7 , d3= 4.5, e3= 1.9;

// valores iniciales de las variables de estado dinámico1
         static double xn= 0.1, yyn= 0.1, zn= 0.1;

/////////////////////////////////////////////////////////////////////////////
   double dinámico_g1(void)
         {
   double hh;

     xn = ((a1 + b1*yyn + c1*zn)*xn + d1*yyn + e1*zn);
     xn = xn - (int)xn;         // eq 16

     yyn = ((a2 + b2*zn + c2*xn)*yyn + d2*zn + e2*xn);
     yyn = yyn - (int)yyn;      // eq 17

     zn = ((a3 + b3*xn + c3*yyn)*zn + d3*xn + e3*yyn);
     zn = zn -(int)zn;          // eq 18

    hh = (xn + yyn + zn);  // con la suma pasa bien
    hh = hh - (int)hh;

    return hh;
         }
/////////////////////////////////////////////////////////////////////////////

         unsigned long dinámico1(void)
         { // llevar a UL el doble de dinámico1
                  unsigned long ww;

                  double dinámico_g1();

                  ww = 4294967295*dinámico_g1();
```

```
                return ww;
        }
//////////////////////////////////////////////////////////////////////////////

        // valores iniciales de dinámico2.
        double aa1 =2.11345, bb1=1.141592653, cc1=1.776666 , dd1= 2.5789123, ee1=
4.98712345;
       double aa2 =3.6987065, bb2=0.312435, cc2=2.19998765 , dd2= 2.2468, ee2=
5.1234567;
   double aa3 =5.909876503, bb3=3.455667788, cc3=1.789789789 , dd3= 3.5654321, ee3=
3.901234567;
// valores iniciales de las variables de estado dinámico2
        static double xxn= 0.1, yyyn= 0.1, zzn= 0.1;
//////////////////////////////////////////////////////////////////////////////
   double dinámico_g2(void)
        {
   double hh;

     xxn = ((aa1 + bb1*yyyn + cc1*zzn)*xxn + dd1*yyyn + ee1*zzn);
     xxn = xxn - (int)xxn;         // eq 16

     yyyn = ((aa2 + bb2*zzn + cc2*xxn)*yyyn + dd2*zzn + ee2*xxn);
     yyyn = yyyn - (int)yyyn;      // eq 17

     zzn = ((aa3 + bb3*xxn + cc3*yyyn)*zzn + dd3*xxn + ee3*yyyn);
     zzn = zn -(int)zn;          // eq 18

    hh = (xxn + yyyn + zzn);  // con la suma pasa bien
    hh = hh - (int)hh;

    return hh;
        }
//////////////////////////////////////////////////////////////////////////////

        unsigned long dinámico2(void)
        {
                unsigned long ww;

                double dinámico_g2();

                ww = 4294967295*dinámico_g2();

                return ww;
        }
```

```
////////////////////////////////////////////////////////////////////////////////

    unsigned long dinámico(void)
          {
       unsigned long w;

                    unsigned long dinámico1();
                    unsigned long dinámico2();

                    w = (dinámico1() + dinámico2());

       return w;
          }

////////////////////////////////////////////////////////////////////////////////



//  programa de prueba
          unsigned long main(void)
{
     int i;
     unsigned long z;

          printf("         prueba dinámico \n");
          for(i=1;i<=10;i++)
          {
                    z = dinámico();
             printf("iter i-> %d numero en main ==> %u \n\n",i, z);
          }

          printf("\n           en 10 iteraciones  x10 es 2859688950   \n");
          printf("iter =  %d el numero en main es ==>   %u \n",i-1,z);

     return (z);
}

////////////////////////////////////////////////////////////////////////////////
```

Resultados Tuftest

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Birthday spacings test: 4096 birthdays, 2^32 days in year
         Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
```

```
Observed    94   336   715   954  1009   775   541   319   169    57    31
(O-E)^2/E   0.1   2.5   0.4   0.5   1.1   0.1   0.8   1.5   2.7   1.3   2.3
            Birthday Spacings: Sum(O-E)^2/E= 13.226, p= 0.789

Euclid's algorithm:
 p-value, steps to gcd:   0.309842
 p-value, dist. of gcd's: 0.163351

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 0.242 0.045 0.976 0.345 0.998 0.355 0.262 0.817
 Bits  8 to 15---> 0.437 0.152 0.780 0.169 0.735 0.846 0.006 0.030
 Bits 16 to 23---> 0.727 0.626 0.234 0.793 0.274 0.052 0.941 0.301
 Bits 24 to 31---> 0.755 0.616 0.726 0.007 0.606 0.431 0.899 0.107
 KS test for the above 32 p values:  0.534
Press any key to continue
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

## Apéndice 3 GNSA criptográficamente resistentes

En esta sección se incluyen los códigos y los resultados de las pruebas con Tuftest a aquellos GNSA_CR, o sea según nuestra calificación criptográficamente resistentes, que han obtenido resultados aceptables con las sucesiones de números seudo aleatorios generados.

### 3.1.- porfiadoUL_CR

#### 3.1.1.- código porfiadoUL_CR

El código en C aparece a continuación.

```
/*

  Este archivo contiene una subrutina porfiadoUL_CR() para implantar porfiadoUL_CR
  que es la version en entero sin signo de 32bits, criptograficamente resistente
  de porfiado4.

  La organizacion es:

  porfiadoUL_CR
          |   // inicializacion
          |---| iniciar_porfiado_CR
          |   | // calentar porfiado_CR y llenar Tabla de mezcla
          |   |--- porfiado4
          |
          |   // generacion
          |---| decimar_porfiado
          |   |   // descartar aleatoriamente los numeros generados, prob= 0.125
          |   |--- porfiado4
          |
          |
          |---| generar_porfiado_CR
          |  |   // extraer al azar Tabla(i,j) y Tabla(j,i), reemplazar valores,
          |  |   // devolver (x+y) - (int)(x+y)
          |  |--- porfiado4
          |       // reinicilizacion aleatoria de porfiado4
          |--- reiniciar_porfiado_CR


  porfiado4 es la version C en doble precisión del algoritmo porfiado bidimensional
   de [4] Lavieri,R. Exploración de nuevas ideas para generación de números pseudo-
aleatorios.
```

```
  Trabajo de ascenso. Universidad Central de Venezuela, Facultad de Ciencias,
  Escuela de Computación. 1984..

  El algoritmo porfiado4 sigue la descripción siguiente:

  Dados x0 = a con a en [100,200]; y0 = b con b en [100,200]; 0< F0 <1; 0< G0 <1;

  A= 151.4751; // constantes de porfiado4
  B= 173.4789;

  Generar recursivamente:

          xn+1 = A/(xn+Fn)
          yn+1 = B/(yn+Gn)

          Fn+1 = DEC(z*yn)
          Gn+1 = DEC(z*xn)

  donde DEC(u)indica la parte decimal del argumento u.

          z = 32 si el computador es binario
          z = 100 si el computador es decimal

  La salida es Fn+1, Gn+1.


  programador R.A. Pastoriza      oct/2007-abril/08
*/

#include<stdio.h>

//       valores iniciales de porfiadoUL_CR
         double x=15.89, y=17.47;
         double Fx= 121.55678, Gy= 1234.34123;

         double A=  151.4751; // constantes de porfiado4
         double B=  173.4789;
//////////////////////////////////////////////////////////////////////////////
        double porfiado4(void)
        {
/*                  printf(" entrando porfiado4 \n");
                    printf(" Fx = %f  \n",Fx);
                    printf(" x = %f  \n", x);
                    printf(" Gy = %f  \n",Gy);
```

```
                printf(" y = %f  \n", y);
                printf(" \n");
*/
            Fx = (100*x) - (int)(100*x);
            Gy = (100*y) - (int)(100*y);

            x = A/(x+Gy);
            y = B/(y+Fx);

/*              printf(" saliendo porfiado4 \n");
                printf(" Fx = %f  \n",Fx);
                printf(" x = %f  \n", x);
                printf(" Gy = %f  \n",Gy);
                printf(" y = %f  \n", y);
                printf(" \n");
*/
            return Fx;
        }
///////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////
  void decimar_porfiado(void)
        {
              double porfiado4();
              double aa, bb;

              int i,ii,j,k,kk,veces;

//                printf(" decimar_porfiado  \n");

              (k = 0, kk = 0);
              (aa = Fx, bb= Gy);

              i = (int)(100*aa);
              j = (int)(100*bb);
              kk = i + j;
              k = (kk)%16;

//                printf(" k = %i  \n", k);
              veces =1;

              if(k>=14) veces = 2; //decimacion 14/16=0.875     2/16=0.125
                                   //decimacion 13/16=0.8125    3/16=0.1875
//                printf(" veces = %i  \n",veces);
```

```
        for(ii=1; ii<veces;ii++)
                {
                  porfiado4();
                }
          }
//////////////////////////////////////////////////////////////////////////

          double Tabla[8][8];
//////////////////////////////////////////////////////////////////////////
    double generar_porfiado_CR(void)
          {// generar un valor
                    int i1,i2;

                    double porfiado4();
                    double u,v;
                    double z;

// generar un valor z con 2 valores extraidos de la tabla de mezcla
                    i1 = (int) 10*Fx;  // indices de extraccion
                    i1 = i1%8;
                    i2 = (int) 10*Gy;
                    i2 = i2%8;

                    u = Tabla[i1][i2]; // valores simetricos extraidos de la Tabla de mezcla
                    v = Tabla[i2][i1];

                    z = (u+v) - (int)(u+v); // valor [0,1] generado

                    porfiado4();

// cambiar seleccion indices

                    Tabla[i1][i2] = Fx;
                    i2 = (i2 + 2)%8; // indices de colocacion asimetricos
                    Tabla[i2][i1] = Gy;

//                  printf(" generar_porfiado_CR z== %f \n",z);

                    return z;
          }

//////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////////////
                    int s=1;
                    int kswitch = 1;

            void reiniciar_porfiado_CR(double z)
                    {
                     double AA=  151.4751;
                     double BB=  173.4789;

//                   int limite=4;
                     int limite= 200000;

                     s = s + (int)(10*z)%10;
//                       printf(" z== %lf  (10*z)== %d \n", z, int (10*z)%10);
//                       printf(" s== %d \n",s);

                     if(s>limite)
                                                {
// inicializa
//                       printf(" *** entrando reiniciar_porfiado_CR *** \n");
//                      printf(" se paso s=  %d \n",s);
                    s= (int)(10*z)%10;
//                      printf(" nuevo inicial s=  %d \n",s);

                    kswitch = (kswitch+1)%2;
// cambia estado
                    switch(kswitch)
                                                 {
                                                 case 0:
                                                         {
                                                                 A = BB;
                                                                 B = AA;
                                                                 break;
                                                         }
                                                 case 1:
                                                         {
                                                                 A = AA;
                                                                 B = BB;
                                                                 break;
                                                         }
                                                 default:
                                                         {
                                                                 break;
                                                         }
```

```
                    }// endswitch

//                  printf("A ==> %f \n",A);
//                  printf("B ==> %f \n",B);
                    }
   }



//////////////////////////////////////////////////////////////////////////////
   int maximo(int a,int b)
       {// maximo de los 2 valores enteros de entrada
           int z;

           if(a<=b) z=b;
           else z=a;

           return z;
       }
//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////
// valores iniciales del calentamiento
           double x0 = 15.32;
           double xy0 = 23.78;

    void iniciar_porfiado_CR(void)

       { // iniciar porfiado
           double x_a;
           double x_b;
           double porfiado4();

    int maximo(int a, int b);
    int k= 51;
   int kk;
   int i,j;

// llenar tabla con 0.0

           for(i=0;i<=7;i++)
           {
               for(j=0;j<=7;j++)
```

```
                {
            Tabla[i][j] = 0.0;
                }
          }

//    calentamiento de porfiado4
//                 printf(" iniciar calentamiento \n");

        (x_a= x0, x_b= xy0);

        kk = (int)(x_a+x_b)&63;
        kk = maximo(k,kk);         // tamanio variable del calentamiento

//      printf(" kk == %d \n",kk);

        for(i=1;i<kk;i++)
            {// calentamiento
                double  hh;
                hh= porfiado4();
            }

  // llenar la tabla inicial

            for(i=0;i<=7;i++)
            {
                for(j=0;j<=7;j++)
                {
                  porfiado4();
                 Tabla[i][j] = Fx;
                 Tabla[j][i] = Gy;
                }
            }

        }

/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////

        int vez_porfiado=1;

    unsigned long porfiadoUL_CR(void)
{// el generador porfiadoUL_CR en si
    unsigned long yy;
```

```
    double z;

        void iniciar_porfiado_CR();
        double generar_porfiado_CR();
        void decimar_porfiado();

        {
                if(vez_porfiado==1)
                {// inicializacion
//                          printf(" entrando iniciar_porfiado \n");

                        iniciar_porfiado_CR();

                        z = generar_porfiado_CR();
//                          printf(" doble en porfiadoUL ==>   %f \n",z);

                       yy = 4294967295*z; // transformacion a entero largo sin signo
//                          printf(" numero en porfiadoUL ==>   %1u \n",yy);

                       vez_porfiado =2;

                       return yy;
                }
      else
                {// generacion
                   decimar_porfiado();

                   z = generar_porfiado_CR();
//                        printf(" doble en porfiadoUL ==>   %f \n",z);

                  yy = 4294967295* z;// transformacion a entero largo sin signo
//                       printf(" numero en porfiado ==>   %u \n",yy);

                 reiniciar_porfiado_CR(z);

//                       printf(" estado s en porfiadoUL ==>   %1u \n",s);

                 return yy;
                }
        }
 }

//////////////////////////////////////////////////////////////////////////////
```

```
//**********************************************************************

////////////////////////////////////////////////////////////////////////////

//  programa de prueba
        unsigned long main(void)
{
    int i;
    unsigned long z;

        printf("        prueba porfiadoUL_CR \n");
        for(i=1;i<=10;i++)
        {
                z = porfiadoUL_CR();
                    printf("iter i-> %d numero en main ==> %u \n\n",i, z);
        }

        printf("\n          en 10 iteraciones  x10 es 2698279694   \n");
        printf("iter =  %d el numero en main es ==>   %u \n",i-1,z);

    return (z);
}

////////////////////////////////////////////////////////////////////////////
```

### 3.1.2.- resultados pruebas Tuftest a porfiadoUL_CR

Los valores resultantes de las pruebas con Tuftest con 100 muestras independientes aparecen en la tabla siguiente.

| | | tuftest | _porfiado_CR | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| # | cumpleaños | gcd Euc | lides | gorila | valor_b | valor_e1 | valor_e2 | valor_g | total valor |
| 1 | 0.102 | 0.8545 | 0.5754 | 0.51 | 2 | 2 | 5 | 5 | 14 |
| 2 | 0.45 | 0.3536 | 0.3716 | 0.37 | 5 | 4 | 4 | 4 | 17 |
| 3 | 0.74 | 0.387 | 0.7179 | 0.935 | 3 | 4 | 3 | 1 | 11 |
| 4 | 0.276 | 0.2038 | 0.16 | 0.042 | 3 | 3 | 2 | 1 | 9 |
| 5 | 0.586 | 0.2632 | 0.2931 | 0.831 | 5 | 3 | 3 | 2 | 13 |
| 6 | 0.472 | 0.5051 | 0.1288 | 0.684 | 5 | 5 | 2 | 4 | 16 |
| 7 | 0.983 | 0.7557 | 0.5254 | 0.684 | 1 | 3 | 5 | 4 | 13 |
| 8 | 0.38 | 0.4953 | 0.8718 | 0.067 | 4 | 5 | 2 | 1 | 12 |
| 9 | 0.79 | 0.7444 | 0.3126 | 0.587 | 3 | 3 | 4 | 5 | 15 |
| 10 | 0.091 | 0.3554 | 0.207 | 0.227 | 1 | 4 | 3 | 3 | 11 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 11 | **0.462** | **0.9249** | **0.2931** | **0.721** | **5** | **1** | **3** | **3** | **12** |
| 12 | **0.875** | **0.519** | **0.2518** | **0.993** | **2** | **5** | **3** | **-1** | **-1** |
| 13 | **0.058** | **0.7248** | **0.192** | **0.267** | **1** | **3** | **2** | **3** | **9** |
| 14 | **0.435** | **0.7572** | **0.6216** | **0.784** | **5** | **3** | **4** | **3** | **15** |
| 15 | **0.14** | **0.3836** | **0.2312** | **0.038** | **2** | **4** | **3** | **1** | **10** |
| 16 | **0.662** | **0.4558** | **0.2719** | **0.43** | **4** | **5** | **3** | **5** | **17** |
| 17 | **0.946** | **0.2452** | **0.7095** | **0.721** | **1** | **3** | **3** | **3** | **10** |
| 18 | **0.861** | **0.7442** | **0.4033** | **0.288** | **2** | **3** | **5** | **3** | **13** |
| 19 | **0.199** | **0.9353** | **0.5188** | **0.17** | **2** | **1** | **5** | **2** | **10** |
| 20 | **0.573** | **0.6362** | **0.9023** | **0.034** | **5** | **4** | **1** | **1** | **11** |
| 21 | **0.203** | **0.8321** | **0.2515** | **0.308** | **3** | **2** | **3** | **4** | **12** |
| 22 | **0.97** | **0.2512** | **0.7349** | **0.268** | **1** | **3** | **3** | **3** | **10** |
| 23 | **0.407** | **0.4954** | **0.6188** | **0.125** | **5** | **5** | **4** | **2** | **16** |
| 24 | **0.799** | **0.4978** | **0.2824** | **0.123** | **3** | **5** | **3** | **2** | **13** |
| 25 | **0.607** | **0.7995** | **0.506** | **0.573** | **4** | **3** | **5** | **5** | **17** |
| 26 | **0.214** | **0.5492** | **0.3637** | **0.595** | **3** | **5** | **4** | **5** | **17** |
| 27 | **0.491** | **0.4336** | **0.0364** | **0.731** | **5** | **5** | **1** | **3** | **14** |
| 28 | **0.7** | **0.8153** | **0.6377** | **0.757** | **3** | **2** | **4** | **3** | **12** |
| 29 | **0.448** | **0.1427** | **0.6721** | **0.85** | **5** | **2** | **4** | **2** | **13** |
| 30 | **0.379** | **0.3347** | **0.5083** | **0.04** | **4** | **4** | **5** | **1** | **14** |
| 31 | **0.41** | **0.2775** | **0.894** | **0.413** | **5** | **3** | **2** | **5** | **15** |
| 32 | **0.23** | **0.332** | **0.2946** | **0.294** | **3** | **4** | **3** | **3** | **13** |
| 33 | **0.228** | **0.2895** | **0.2119** | **0.513** | **3** | **3** | **3** | **5** | **14** |
| 34 | **0.915** | **0.5693** | **0.2756** | **0.24** | **1** | **5** | **3** | **3** | **12** |
| 35 | **0.8** | **0.2672** | **0.1247** | **0.5** | **2** | **3** | **2** | **6** | **13** |
| 36 | **0.934** | **0.1617** | **0.0329** | **0.792** | **1** | **2** | **1** | **3** | **7** |
| 37 | **0.969** | **0.5608** | **0.4726** | **0.11** | **1** | **5** | **5** | **2** | **13** |
| 38 | **0.811** | **0.1672** | **0.5889** | **0.347** | **2** | **2** | **5** | **4** | **13** |
| 39 | **0.927** | **0.337** | **0.6008** | **0.392** | **1** | **4** | **4** | **4** | **13** |
| 40 | **0.142** | **0.6146** | **0.5029** | **0.76** | **2** | **4** | **5** | **3** | **14** |
| 41 | **0.516** | **0.3983** | **0.1218** | **0.869** | **5** | **4** | **2** | **2** | **13** |
| 42 | **0.761** | **0.2333** | **0.5834** | **0.39** | **3** | **3** | **5** | **4** | **15** |
| 43 | **0.661** | **0.5088** | **0.7095** | **0.016** | **4** | **5** | **3** | **1** | **13** |
| 44 | **0.057** | **0.7017** | **0.3227** | **0.093** | **1** | **3** | **4** | **1** | **9** |
| 45 | **0.488** | **0.6346** | **0.267** | **0.385** | **5** | **4** | **3** | **4** | **16** |
| 46 | **0.109** | **0.3308** | **0.419** | **0.337** | **2** | **4** | **5** | **4** | **15** |
| 47 | **0.854** | **0.553** | **0.6303** | **0.056** | **2** | **5** | **4** | **1** | **12** |
| 48 | **0.15** | **0.4333** | **0.3894** | **0.728** | **2** | **5** | **4** | **3** | **14** |
| 49 | **0.291** | **0.6514** | **0.5023** | **0.122** | **3** | **4** | **5** | **2** | **14** |
| 50 | **0.421** | **0.2682** | **0.7095** | **0.227** | **5** | **3** | **3** | **3** | **14** |
| 51 | **0.976** | **0.159** | **0.6843** | **0.258** | **1** | **2** | **4** | **3** | **10** |
| 52 | **0.182** | **0.587** | **0.4542** | **0.725** | **2** | **5** | **5** | **3** | **15** |
| 53 | **0.172** | **0.4461** | **0.3319** | **0.689** | **2** | **5** | **4** | **4** | **15** |
| 54 | **0.645** | **0.6228** | **0.1014** | **0.793** | **4** | **4** | **2** | **3** | **13** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 55 | **0.35** | **0.5122** | **0.422** | **0.867** | **4** | **5** | **5** | **2** | **16** |
| 56 | **0.408** | **0.9484** | **0.5799** | **0.037** | **5** | **1** | **5** | **1** | **12** |
| 57 | **0.216** | **0.5196** | **0.6845** | **0.843** | **3** | **5** | **4** | **2** | **14** |
| 58 | **0.55** | **0.7359** | **0.351** | **0.203** | **5** | **3** | **4** | **3** | **15** |
| 59 | **0.591** | **0.2386** | **0.5009** | **0.694** | **5** | **3** | **5** | **4** | **17** |
| 60 | **0.523** | **0.2007** | **0.4573** | **0.411** | **5** | **3** | **5** | **5** | **18** |
| 61 | **0.843** | **0.4998** | **0.1553** | **0.78** | **2** | **5** | **2** | **3** | **12** |
| 62 | **0.421** | **0.8682** | **0.7089** | **0.975** | **5** | **2** | **3** | **1** | **11** |
| 63 | **0.916** | **0.1014** | **0.8162** | **0.711** | **1** | **2** | **2** | **3** | **8** |
| 64 | **0.558** | **0.2906** | **0.8244** | **0.456** | **5** | **3** | **2** | **5** | **15** |
| 65 | **0.989** | **0.1367** | **0.8276** | **0.423** | **1** | **2** | **2** | **5** | **10** |
| 66 | **0.596** | **0.5826** | **0.4274** | **0.151** | **5** | **5** | **5** | **2** | **17** |
| 67 | **0.196** | **0.5033** | **0.7448** | **0.833** | **2** | **5** | **3** | **2** | **12** |
| 68 | **0.508** | **0.2945** | **0.284** | **0.374** | **5** | **3** | **3** | **4** | **15** |
| 69 | **0.555** | **0.4465** | **0.4113** | **0.84** | **5** | **5** | **5** | **2** | **17** |
| 70 | **0.721** | **0.8202** | **0.3502** | **0.46** | **3** | **2** | **4** | **5** | **14** |
| 71 | **0.031** | **0.2472** | **0.5556** | **0.47** | **1** | **3** | **5** | **5** | **14** |
| 72 | **0.896** | **0.2407** | **0.9006** | **0.83** | **2** | **3** | **1** | **2** | **8** |
| 73 | **0.779** | **0.1365** | **0.3377** | **0.948** | **3** | **2** | **4** | **1** | **10** |
| 74 | **0.981** | **0.9256** | **0.3735** | **0.234** | **1** | **1** | **4** | **3** | **9** |
| 75 | **0.879** | **0.4838** | **0.686** | **0.543** | **2** | **5** | **4** | **5** | **16** |
| 76 | **0.284** | **0.4287** | **0.5872** | **0.033** | **3** | **5** | **5** | **1** | **14** |
| 77 | **0.108** | **0.9012** | **0.3657** | **0.999** | **2** | **1** | **4** | **-1** | **-1** |
| 78 | **0.105** | **0.1988** | **0.6062** | **0.505** | **2** | **2** | **4** | **5** | **13** |
| 79 | **0.308** | **0.5879** | **0.546** | **0.35** | **4** | **5** | **5** | **4** | **18** |
| 80 | **0.714** | **0.6867** | **0.4487** | **0.385** | **3** | **4** | **5** | **4** | **16** |
| 81 | **0.623** | **0.5402** | **0.2742** | **0.107** | **4** | **5** | **3** | **2** | **14** |
| 82 | **0.457** | **0.4701** | **0.7549** | **0.745** | **5** | **5** | **3** | **3** | **16** |
| 83 | **0.828** | **0.5554** | **0.7815** | **0.803** | **2** | **5** | **3** | **2** | **12** |
| 84 | **0.982** | **0.2582** | **0.8865** | **0.991** | **1** | **3** | **2** | **-1** | **-1** |
| 85 | **0.857** | **0.5679** | **0.4398** | **0.176** | **2** | **5** | **5** | **2** | **14** |
| 86 | **0.114** | **0.275** | **0.262** | **0.187** | **2** | **3** | **3** | **2** | **10** |
| 87 | **0.47** | **0.6331** | **0.3912** | **0.594** | **5** | **4** | **4** | **5** | **18** |
| 88 | **0.498** | **0.536** | **0.1212** | **0.364** | **5** | **5** | **2** | **4** | **16** |
| 89 | **0.482** | **0.4252** | **0.2164** | **0.233** | **5** | **5** | **3** | **3** | **16** |
| 90 | **0.768** | **0.8201** | **0.5993** | **0.461** | **3** | **2** | **5** | **5** | **15** |
| 91 | **0.382** | **0.8803** | **0.3964** | **0.796** | **4** | **2** | **4** | **3** | **13** |
| 92 | **0.201** | **0.2168** | **0.6316** | **0.06** | **3** | **3** | **4** | **1** | **11** |
| 93 | **0.463** | **0.9645** | **0.1254** | **0.791** | **5** | **1** | **2** | **3** | **11** |
| 94 | **0.832** | **0.1497** | **0.7378** | **0.423** | **2** | **2** | **3** | **5** | **12** |
| 95 | **0.322** | **0.8781** | **0.7794** | **0.271** | **4** | **2** | **3** | **3** | **12** |
| 96 | **0.228** | **0.9332** | **0.3053** | **0.67** | **3** | **1** | **4** | **4** | **12** |
| 97 | **0.647** | **0.7312** | **0.724** | **0.728** | **4** | **3** | **3** | **3** | **13** |
| 98 | **0.724** | **0.6023** | **0.2052** | **0.395** | **3** | **4** | **3** | **4** | **14** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 99 | **0.346** | **0.0565** | **0.4508** | **0.484** | **4** | **1** | **5** | **5** | **15** |
| 100 | **0.888** | **0.7267** | **0.174** | **0.095** | **2** | **3** | **2** | **1** | **8** |
| | **===> final** | **tuftest <===** | | | | | | | |

## 3.2.- Kiss_me

### 3.2.1.- código Kiss_me

El código en C aparece a continuación.

```
#include <stdio.h>
/*

  Este archivo contiene una subrutina Kiss99_me() para implantar un generador
  que es la version en entero sin signo de 32bits, criptográficamente resistente
  de Kiss99.

  Kiss_me es un GNSA_CR que se compone de 2 instancias distintas de Kiss99,
  con un calentamiento variable en Kiss99_o y Kiss99_b, una tabla de mezcla
  y la mezcla de los 2 números extraídos de la tabla. Adicionalmente tiene una
  reinicialización variable.

  Kiss99_o = (CONG xor MWC) + SHR3 donde:

  CONG: x == 69069*x+ 1234567 (mod 2**32)

  MWC: zw = ((z<<16)+ w)
          z = 36969*(z&65535)+(z>>16)
          w = 18000*(w&65535)+(w>>16)

  SHR3:(jsr^=(jsr<<17) + 123, jsr^=(jsr>>13)  + 456, jsr^=(jsr<<5)  + 789)

y

  Kiss99_b = (CONG xor MWC) + SHR3 donde:

  CONG: x == 1566083941*x+ 1234567 (mod 2**32)

  MWC: zw = ((z<<16)+ w)
          z = 36969*(z&65535)+(z>>16)
          w = 18000*(w&65535)+(w>>16)

  SHR3:(jsr^=(jsr<<16) + 123, jsr^=(jsr>>7)  + 456, jsr^=(jsr<<11)  + 789)


  La organización de Kiss_me es:
```

```
Kiss99_me
    |    // inicializacion
    |---| init_Kiss99_dual
    |   | // calentar Kiss_me y llenar Tabla de mezcla
    |   |--- init_Kiss99_o
    |   |--- init Kiss99_b
    |   |--- Kiss99_o
    |   |--- Kiss99_b
    |
    |    // generar valores
    |---| Kiss99_dual
    |  |  // extraer al azar Tabla(i,j) y Tabla(j,i), reemplazar valores,
    |  |  // y devolver valor
    |  |--- Kiss99_o
    |  |--- Kiss99_b
    |
    |    // reinicilizacion aleatoria de Kiss_me
    |--- reiniciar_Kiss99_me
```

Kiss99 es la version C en doble precision del algoritmo Kiss de [6] Marsaglia, en Random Numbers for C: End, at last? From: George Marsaglia <geo@stat.fsu.edu>
Date: Thu, 21 Jan 1999 03:08:52 GMT
Organization: Florida State University

El algoritmo Kiss99 sigue la descripcion siguiente:

The KISS generator, (Keep It Simple Stupid), is
designed to combine the two multiply-with-carry
generators in MWC with the 3-shift register SHR3 and
the congruential generator CONG, using addition and
exclusive-or. Period about 2^123.
It is one of my favorite generators.

The MWC generator concatenates two 16-bit multiply-
with-carry generators, x(n)=36969x(n-1)+carry,
y(n)=18000y(n-1)+carry mod 2^16, has period about
2^60 and seems to pass all tests of randomness. A
favorite stand-alone generator---faster than KISS,
which contains it.

SHR3 is a 3-shift-register generator with period
2^32-1. It uses y(n)=y(n-1)(I+L^17)(I+R^13)(I+L^5),

```
  with the y's viewed as binary vectors, L the 32x32
  binary matrix that shifts a vector left 1, and R its
  transpose. SHR3 seems to pass all except those
  related to the binary rank test, since 32 successive
  values, as binary vectors, must be linearly
  independent, while 32 successive truly random 32-bit
  integers, viewed as binary vectors, will be linearly
  independent only about 29% of the time.

  CONG is a congruential generator with the widely used 69069
  multiplier: x(n)=69069x(n-1)+1234567. It has period
  2^32. The leading half of its 32 bits seem to pass
  tests, but bits in the last half are too regular.

  programador R.A. Pastoriza       oct/2007-abril/08
*/
// inicializacion de Kiss99_o  y Kiss99_b
//          unsigned long h1=6171437,h2=314151721,h3=415981617,h4=7863119;
//                                        unsigned                          long
r1=1234567,r2=141516321,r3=3435362121,r4=9988774545;
///////////////////////////////////////////////////////////////////////////////////

               unsigned long x_o,jsr_o,z_o,w_o;
///////////////////////////////////////////////////////////////////////////////////
        void init_Kiss99_o(x1,x2,x3,x4)
                     unsigned long x1,x2,x3,x4;
                     {// inicializacion de Kiss99_o con calentamiento
                     unsigned long Kiss99_o();
        int maximo();

                     int k= 31;
                     int kk;
                     int i;

                     (x_o= x1, jsr_o= x2, z_o= x3, w_o= x4);

                     kk = (x_o+jsr_o+z_o+w_o)&63;
                     kk = maximo(k,kk);          // tamanio variable del calentamiento

//                   printf(" kk es --> %i \n", kk);

                     for(i=1;i<kk;i++)
                            {// calentamiento
                             unsigned long zz;
```

```
                        zz = Kiss99_o();
//                      printf(" calentando   %u \n",zz);
                       }
                }
/////////////////////////////////////////////////////////////////////////////////////
    int maximo(a,b)
                int a,b;
          {// maximo de los 2 valores enteros de entrada
                int z;

                if(a<=b) z=b;
                else z=a;

                return z;
          }
/////////////////////////////////////////////////////////////////////////////////////
         unsigned long Kiss99_o(void)                    // coeficientes en Kiss; en Random
Numbers for C: End, at last?
                { // Kiss99 original
            unsigned long y_o;

      x_o = 69069*x_o+1234567;                 // el congruencial

          jsr_o ^= (jsr_o<<13) + 123;                     // el SHR3 original mas terminos
independientes
      jsr_o ^= (jsr_o>>17)  + 456;
      jsr_o ^= (jsr_o<<5) + 789;

      z_o=36969*(z_o&65535)+(z_o>>16) + 34567;    // los MWC originales mas terminos
independientes
      w_o=18000*(w_o&65535)+(w_o>>16) + 76543;
            y_o= (z_o<<16)+w_o;

      return ((x_o^y_o)+jsr_o);               // valor final
                }

/////////////////////////////////////////////////////////////////////////////////////

// Kiss99_b con inicializacion separada y GCL y terminos independientes distintos
                unsigned long x_b,jsr_b,z_b,w_b;

       void init_Kiss99_b(x1,x2,x3,x4)
                unsigned long x1,x2,x3,x4;
          {// inicializacion Kiss99_b con calentamiento
```

```
                unsigned long Kiss99_b();
        int maximo();

                int k= 27;
                int kk;
                int i;

                (x_b= x1, jsr_b=  x2, z_b= x3, w_b= x4);

                kk = (x_b+jsr_b+z_b+w_b)&63;
                kk = maximo(k,kk);          // tamanio variable del calentamiento

                for(i=1;i<kk;i++)
                {// calentamiento
                        unsigned long hh;
                        hh= Kiss99_b();
                }
        }
////////////////////////////////////////////////////////////////////////////////////////////
        unsigned long Kiss99_b(void)       // otros coeficientes en Kiss; en Random Numbers
for C: End, at last?
                { // otra version de Kiss99
            unsigned long y_b;

            x_b = 1566083941*x_b + 1234567;              //  otro GCL

        jsr_b ^= (jsr_b<<16) + 123;                  // el SHR3  (16,-7,11) de Pres y otros pag
1143 mas terminos independientes
        jsr_b ^= (jsr_b>>7)  + 456;
        jsr_b ^= (jsr_b<<11) + 789;

            z_b=36969*(z_b&65535)+(z_b>>16)  +  34567;        //  los  MWC  originales  mas
terminos independientes
        w_b=18000*(w_b&65535)+(w_b>>16) + 76543;
            y_b= (z_b<<16)+w_b;

        return ((x_b^y_b)+jsr_b);                  // valor final
                }

////////////////////////////////////////////////////////////////////////////////////////////
            unsigned long Tabla[64];

            void init_Kiss_dual(h1,h2,h3,h4,r1,r2,r3,r4)
                {//inicializacion de Kiss99_dual
```

```
                int i;
                void init_Kiss99_o();
                void init_KIss99_b();

        init_Kiss99_o(h1,h2,h3,h4);
        init_Kiss99_b(r1,r2,r3,r4);

        for(i=0;i<=62;i= i +2)
                    {// inicializar la Tabla alternando la inclusion de Kiss_o y Kiss99_b
en la tabla de mezcla
                      Tabla[i]   = Kiss99_o();
          Tabla[i+1] = Kiss99_b();
                    }
               }
////////////////////////////////////////////////////////////////////////////////////////
       unsigned long Tabla[64];

               unsigned long Kiss_dual()
               { //  generacion de los numeros en Kiss99_dual
               unsigned long Kiss99_o();
               unsigned long Kiss99_b();

               unsigned long vv1,vv2;
               unsigned long w1,w2;
               unsigned long salir;

               int i,j,k;
               int indice1,indice2;

               vv1 = Kiss99_o();
               indice1= vv1%64;
               vv2 = Kiss99_b();
       indice2= vv2%64;


               w1 = Tabla[indice2];
               Tabla[indice2]= vv1;
               w2 = Tabla[indice1];
               Tabla[indice1] =vv2;
               salir= (w1+w2);

               return(salir);
          }
```

```
/////////////////////////////////////////////////////////////////////////////////

            int s=0;

            unsigned long reiniciar_Kiss_me(unsigned long w)
            {// reinicar Kiss99_me
             unsigned long a_o_v,b_o_v,c_o_v,d_o_v;

             unsigned long a_b_v,b_b_v,c_b_v,d_b_v;

//           int limite=200;
             int limite= 400000;

             s = s + (w>>12)%10;
//              printf(" w== %u  (w>12)== %u \n", w, (w>12)%10);
//              printf(" s== %d \n",s);

             if(s>limite)
                              {
// inicializa
//                         printf(" *** entrando reiniciar_Kiss99_me *** \n");
//             printf(" se paso s=  %d \n",s);
            s= (w>>12)%10;
//             printf(" nuevo inicial s=  %d \n",s);
// guarda actuales
                 a_o_v=  x_o;
                 b_o_v=  jsr_o;
                 c_o_v=  z_o;
                 d_o_v=  w_o;

                a_b_v=  x_b;
                 b_b_v=  jsr_b;
                 c_b_v=  z_b;
                 d_b_v=  w_b;
// cambia estado a,b por c,d y viceversa

                     x_o  = a_b_v;
                     jsr_o= b_b_v;
                     z_o  = c_b_v;
                     w_o  = d_b_v;

                     x_b  = a_o_v;
                     jsr_b= b_o_v;
                     z_b  = c_o_v;
```

```
                        w_b  = d_o_v;

//                                      printf("a_o_v==> %u \n",a_o_v);
//                                      printf("x_b==> %u \n",a_o_v);
                                          }
                return s;
        }

/////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////
                int vez_dual=1;
    unsigned long Kiss_me(void)
{
    unsigned long yy;

        void init_Kiss_dual();
        unsigned long Kiss_dual();
        {
                if(vez_dual==1)
                {// inicializacion
                        init_Kiss_dual(h1,h2,h3,h4,r1,r2,r3,r4);
                        vez_dual =2;
        yy = Kiss_dual(); // extraer valor

                        return yy;
                }
      else
                {// generacion
        yy = Kiss_dual(); // extraer valor
        reiniciar_Kiss_me(yy);

       return yy;
                }
        }
 }
/////////////////////////////////////////////////////////////////////////////

//  programa de prueba
        unsigned long main(void)
{
    int i;
    unsigned long z;
```

```
        printf("          prueba Kiss99_me \n");
        for(i=1;i<=10;i++)
        {
           z = Kiss99_me();

           printf("iter i-> %d numero en main ==> %u \n\n",i, z);
        }

        printf("\n            en 10 iteraciones  x10 es 3422932408  \n");
        printf("iter =  %d el numero en main es ==>   %u \n",i-1,z);

    return (z);
}

//////////////////////////////////////////////////////////////////////////////
```

### 3.2.2.- resultados pruebas Tuftest a Kiss_me

Los valores resultantes de las pruebas con Tuftest con 100 muestras independientes aparecen en la tabla siguiente.

| | | tuftest | _Kiss_me | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **cumpleaños** | **gcd Euc** | **lides** | **gorila** | **valor_b** | **valor_e1** | **valor_e2** | **valor_g** | **total valor** |
| 1 | **0.166** | **0.7953** | **0.2223** | **-4.639** | **2** | **3** | **3** | **-45** | **-1** |
| 2 | **0.834** | **0.3902** | **0.7264** | **0.174** | **2** | **4** | **3** | **2** | **11** |
| 3 | **0.169** | **0.6632** | **0.905** | **0.536** | **2** | **4** | **1** | **5** | **12** |
| 4 | **0.907** | **0.1867** | **0.4804** | **-6.094** | **1** | **2** | **5** | **-59** | **-1** |
| 5 | **0.965** | **0.4882** | **0.2041** | **0.041** | **1** | **5** | **3** | **1** | **10** |
| 6 | **0.265** | **0.2217** | **0.5644** | **0.006** | **3** | **3** | **5** | **-1** | **-1** |
| 7 | **0.744** | **0.3243** | **0.7221** | **-4.393** | **3** | **4** | **3** | **-42** | **-1** |
| 8 | **0.848** | **0.2252** | **0.6694** | **0.225** | **2** | **3** | **4** | **3** | **12** |
| 9 | **0.600** | **0.8703** | **0.6846** | **0.629** | **4** | **2** | **4** | **4** | **14** |
| 10 | **0.370** | **0.4726** | **0.7193** | **-7.771** | **4** | **5** | **3** | **-76** | **-1** |
| 11 | **0.672** | **0.3662** | **0.1723** | **0.11** | **4** | **4** | **2** | **2** | **12** |
| 12 | **0.387** | **0.4141** | **0.87** | **0.43** | **4** | **5** | **2** | **5** | **16** |
| 13 | **0.703** | **0.2516** | **0.8754** | **0.265** | **3** | **3** | **2** | **3** | **11** |
| 14 | **0.267** | **0.6267** | **0.0914** | **-6.27** | **3** | **4** | **1** | **-61** | **-1** |
| 15 | **0.044** | **0.3564** | **0.4759** | **0.578** | **1** | **4** | **5** | **5** | **15** |
| 16 | **0.267** | **0.5108** | **0.4598** | **0.242** | **3** | **5** | **5** | **3** | **16** |
| 17 | **0.574** | **0.5331** | **0.5027** | **-7.159** | **5** | **5** | **5** | **-70** | **-1** |
| 18 | **0.160** | **0.7787** | **0.8397** | **0.035** | **2** | **3** | **2** | **1** | **8** |
| 19 | **0.203** | **0.8177** | **0.6858** | **0.295** | **3** | **2** | **4** | **3** | **12** |
| 20 | **0.131** | **0.3861** | **0.2171** | **0.585** | **2** | **4** | **3** | **5** | **14** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 21 | **0.943** | **0.5926** | **0.6867** | **0.32** | **1** | **5** | **4** | **4** | **14** |
| 22 | **0.963** | **0.4124** | **0.0263** | **0.346** | **1** | **5** | **1** | **4** | **11** |
| 23 | **0.023** | **0.1832** | **0.3375** | **0.641** | **1** | **2** | **4** | **4** | **11** |
| 24 | **0.792** | **0.4577** | **0.6773** | **-4.854** | **3** | **5** | **4** | **-47** | **-1** |
| 25 | **0.517** | **0.8583** | **0.4717** | **0.488** | **5** | **2** | **5** | **5** | **17** |
| 26 | **0.438** | **0.7504** | **0.6971** | **0.557** | **5** | **3** | **4** | **5** | **17** |
| 27 | **0.896** | **0.6416** | **0.8385** | **0.46** | **2** | **4** | **2** | **5** | **13** |
| 28 | **0.915** | **0.701** | **0.647** | **0.13** | **1** | **3** | **4** | **2** | **10** |
| 29 | **0.163** | **0.3347** | **0.5495** | **-8.518** | **2** | **4** | **5** | **-84** | **-1** |
| 30 | **0.905** | **0.2786** | **0.8978** | **-4.398** | **1** | **3** | **2** | **-42** | **-1** |
| 31 | **0.268** | **0.4593** | **0.3446** | **0.04** | **3** | **5** | **4** | **1** | **13** |
| 32 | **0.899** | **0.4757** | **0.8456** | **-5.484** | **2** | **5** | **2** | **-53** | **-1** |
| 33 | **0.487** | **0.3163** | **0.8457** | **-6.535** | **5** | **4** | **2** | **-64** | **-1** |
| 34 | **0.702** | **0.3923** | **0.8089** | **-4.773** | **3** | **4** | **2** | **-46** | **-1** |
| 35 | **0.533** | **0.6286** | **0.4004** | **0.594** | **5** | **4** | **5** | **5** | **19** |
| 36 | **0.912** | **0.5865** | **0.5356** | **0.531** | **1** | **5** | **5** | **5** | **16** |
| 37 | **0.285** | **0.7016** | **0.772** | **0.589** | **3** | **3** | **3** | **5** | **14** |
| 38 | **0.061** | **0.2916** | **0.1847** | **-5.837** | **1** | **3** | **2** | **-57** | **-1** |
| 39 | **0.476** | **0.6448** | **0.438** | **0.621** | **5** | **4** | **5** | **4** | **18** |
| 40 | **0.228** | **0.8513** | **0.1651** | **-9.13** | **3** | **2** | **2** | **-90** | **-1** |
| 41 | **0.897** | **0.4444** | **0.2838** | **-7.871** | **2** | **5** | **3** | **-77** | **-1** |
| 42 | **0.255** | **0.5113** | **0.9432** | **-4.778** | **3** | **5** | **1** | **-46** | **-1** |
| 43 | **0.958** | **0.2201** | **0.9463** | **-5.241** | **1** | **3** | **1** | **-51** | **-1** |
| 44 | **0.990** | **0.736** | **0.5839** | **0.616** | **1** | **3** | **5** | **4** | **13** |
| 45 | **0.220** | **0.3149** | **0.67** | **0.42** | **3** | **4** | **4** | **5** | **16** |
| 46 | **0.786** | **0.3207** | **0.3199** | **0.102** | **3** | **4** | **4** | **2** | **13** |
| 47 | **0.092** | **0.4615** | **0.5967** | **0.247** | **1** | **5** | **5** | **3** | **14** |
| 48 | **0.744** | **0.3312** | **0.5698** | **-6.024** | **3** | **4** | **5** | **-59** | **-1** |
| 49 | **0.939** | **0.9938** | **0.7678** | **-4.313** | **1** | **-1** | **3** | **-42** | **-1** |
| 50 | **0.356** | **0.6173** | **0.1935** | **0.446** | **4** | **4** | **2** | **5** | **15** |
| 51 | **0.151** | **0.7394** | **0.5789** | **-4.137** | **2** | **3** | **5** | **-40** | **-1** |
| 52 | **0.803** | **0.8449** | **0.5218** | **0.599** | **2** | **2** | **5** | **5** | **14** |
| 53 | **0.650** | **0.6603** | **0.8784** | **-5.954** | **4** | **4** | **2** | **-58** | **-1** |
| 54 | **0.654** | **0.4963** | **0.7531** | **-6.19** | **4** | **5** | **3** | **-60** | **-1** |
| 55 | **0.826** | **0.5721** | **0.343** | **0.486** | **2** | **5** | **4** | **5** | **16** |
| 56 | **0.089** | **0.6206** | **0.595** | **-4.104** | **1** | **4** | **5** | **-40** | **-1** |
| 57 | **0.311** | **0.3882** | **0.8146** | **-7.309** | **4** | **4** | **2** | **-72** | **-1** |
| 58 | **0.808** | **0.502** | **0.6413** | **-4.611** | **2** | **5** | **4** | **-45** | **-1** |
| 59 | **0.933** | **0.7691** | **0.2499** | **-5.025** | **1** | **3** | **3** | **-49** | **-1** |
| 60 | **0.752** | **0.9193** | **0.6689** | **-4.346** | **3** | **1** | **4** | **-42** | **-1** |
| 61 | **0.874** | **0.6559** | **0.7146** | **0.575** | **2** | **4** | **3** | **5** | **14** |
| 62 | **0.150** | **0.3015** | **0.6186** | **-5.819** | **2** | **4** | **4** | **-57** | **-1** |
| 63 | **0.798** | **0.7473** | **0.3011** | **-4.441** | **3** | **3** | **4** | **-43** | **-1** |
| 64 | **0.781** | **0.4114** | **0.5257** | **-7.655** | **3** | **5** | **5** | **-75** | **-1** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 65 | **0.037** | **0.3099** | **0.7407** | **0.284** | **1** | **4** | **3** | **3** | **11** |
| 66 | **0.566** | **0.667** | **0.3197** | **0.156** | **5** | **4** | **4** | **2** | **15** |
| 67 | **0.576** | **0.5383** | **0.7126** | **-4.251** | **5** | **5** | **3** | **-41** | **-1** |
| 68 | **0.209** | **0.6269** | **0.9349** | **-5.04** | **3** | **4** | **1** | **-49** | **-1** |
| 69 | **0.192** | **0.5322** | **0.215** | **0.088** | **2** | **5** | **3** | **1** | **11** |
| 70 | **0.016** | **0.081** | **0.5259** | **-6.128** | **1** | **1** | **5** | **-60** | **-1** |
| 71 | **0.074** | **0.5521** | **0.5134** | **0.157** | **1** | **5** | **5** | **2** | **13** |
| 72 | **0.677** | **0.5765** | **0.5574** | **0.225** | **4** | **5** | **5** | **3** | **17** |
| 73 | **0.241** | **0.7791** | **0.5565** | **0.63** | **3** | **3** | **5** | **4** | **15** |
| 74 | **0.765** | **0.0367** | **0.524** | **0.373** | **3** | **1** | **5** | **4** | **13** |
| 75 | **0.688** | **0.7368** | **0.6392** | **0.468** | **4** | **3** | **4** | **5** | **16** |
| 76 | **0.178** | **0.5941** | **0.7733** | **0.545** | **2** | **5** | **3** | **5** | **15** |
| 77 | **0.650** | **0.3603** | **0.2031** | **0.243** | **4** | **4** | **3** | **3** | **14** |
| 78 | **0.963** | **0.7831** | **0.4648** | **0.358** | **1** | **3** | **5** | **4** | **13** |
| 79 | **0.644** | **0.5382** | **0.1302** | **0.154** | **4** | **5** | **2** | **2** | **13** |
| 80 | **0.465** | **0.5919** | **0.4835** | **0.21** | **5** | **5** | **5** | **3** | **18** |
| 81 | **0.623** | **0.4888** | **0.8032** | **0.049** | **4** | **5** | **2** | **1** | **12** |
| 82 | **0.404** | **0.3489** | **0.8667** | **0.588** | **5** | **4** | **2** | **5** | **16** |
| 83 | **0.617** | **0.4285** | **0.5053** | **0.404** | **4** | **5** | **5** | **5** | **19** |
| 84 | **0.631** | **0.4137** | **0.6326** | **0.356** | **4** | **5** | **4** | **4** | **17** |
| 85 | **0.634** | **0.1583** | **0.2366** | **-4.272** | **4** | **2** | **3** | **-41** | **-1** |
| 86 | **0.289** | **0.3591** | **0.4857** | **0.432** | **3** | **4** | **5** | **5** | **17** |
| 87 | **0.764** | **0.7102** | **0.2656** | **0.372** | **3** | **3** | **3** | **4** | **13** |
| 88 | **0.880** | **0.4695** | **0.7574** | **0.214** | **2** | **5** | **3** | **3** | **13** |
| 89 | **0.228** | **0.1978** | **0.0932** | **0.211** | **3** | **2** | **1** | **3** | **9** |
| 90 | **0.530** | **0.2858** | **0.1696** | **0.089** | **5** | **3** | **2** | **1** | **11** |
| 91 | **0.707** | **0.3666** | **0.1778** | **-4.925** | **3** | **4** | **2** | **-48** | **-1** |
| 92 | **0.253** | **0.1589** | **0.7482** | **-5.52** | **3** | **2** | **3** | **-54** | **-1** |
| 93 | **0.163** | **0.264** | **0.2765** | **0.007** | **2** | **3** | **3** | **-1** | **-1** |
| 94 | **0.929** | **0.1811** | **0.5394** | **-5.893** | **1** | **2** | **5** | **-57** | **-1** |
| 95 | **0.111** | **0.7426** | **0.9467** | **0.49** | **2** | **3** | **1** | **5** | **11** |
| 96 | **0.710** | **0.4418** | **0.8974** | **0.319** | **3** | **5** | **2** | **4** | **14** |
| 97 | **0.431** | **0.4683** | **0.2944** | **0.458** | **5** | **5** | **3** | **5** | **18** |
| **98** | **0.652** | **0.6343** | **0.4363** | **0.261** | **4** | **4** | **5** | **3** | **16** |

## 3.3.- girar_me

### 3.3.1.- código girar_me

El código en C aparece a continuación.

```
# include <stdio.h>
/*

girar_me es un GNSA_CR construido sobre la base de girar_dec para implantar un generador
que es la version en entero sin signo de 32bits, criptograficamente resistente.
Incluye dos instancias distintas girar_dec1 y girar_dec2.

Cada instancia de girar_dec incluye su propia inicializacion y decimacion.

Estas instancias se emplean para llenar una tabla de mezcla en valores doble precision
alternando los 2 valores generados por girar_dec1 y girar_dec2. La conversion a entero
sin signo se hace despues de extraer el valor de la tabla.

  La organización es:

  girar_me_me
           |    // inicializacion
           |---| init_girar_me
           |   | // calentar girar_me y llenar Tabla de mezcla
           |   |--- init_girar_dec1
           |   |--- init girar_dec2
           |   |--- girar_dec1
           |   |--- girar_dec2
           |   |--- calcular_doble
           |
           |    // generar valores
           |---| girar_me
           |    |   // extraer al azar Tabla(i,j) y Tabla(j,i), reemplazar valores,
           |    |      y devolver valor doble
           |    |--- girar_dec1
           |    |--- girar_dec2
           |    |--- calcular_doble
           |
           |     // reinicializacion aleatoria de Kiss_me
           |--- reiniciar_girar_me
```

```
  girar_dec1 contiene el generador primario, y su decimación:

  entrada:
        a1,a2,a3,a4
  calcular
            a1 = ~(a1^(a1<<5) + a2 + 1234567);
                a2 = a2^(a2>>13) + a1 + 76543;

                a3 = a3^(a3<<6)+ a4 + 314151;
                a4 = ~(a4^(a4>>7)+ a3 + 349765);

                z = (a1^a3) + (a2^a4);
  salida:
        z

  girar_dec2 contiene el generador primario girar, con otros términos independientes,
  y su decimación:

  entrada:
        b1, b2, b3, b4
  calcular
        b1 = ~(b1^(b1<<5) + b2 + 7654321);
        b2 = b2^(b2>>13) + b1 + 1234567;

        b3 = b3^(b3<<6)+ b4 + 151413;
       b4 = ~(b4^(b4>>7)+ b3 + 567943);

        z = (b1+b2+b3+b4);
  salida:
        z

*/

// valores iniciales Ul de girar_dec1 y girar_dec2
unsigned long ii1=6171437, ii2=314151721, ii3=415981617,  ii4=7863119;
unsigned long jj1=1234567, jj2=141516321, jj3=3435362121, jj4=1988774545;

////////////////////////////////////////////////////////////////////////////////////////

    int maximo(int a,int b)
          {// maximo de los 2 valores enteros de entrada
                int z;
```

```
            if(a<=b) z=b;
            else z=a;

            return z;
        }

/////////////////////////////////////////////////////////////////////////////////////
   unsigned long a1,a2,a3,a4;
        unsigned long b1,b2,b3,b4;

  void init_girar_dec1(unsigned long x1,unsigned long x2,unsigned long x3,unsigned long
x4)

  {
   int k=15;
   int kk;
        int i;

        unsigned long girar_dec1();

   (a1=x1,a2=x2,a3=x3,a4=x4);

        kk = (a1+a2+a3+a4)&63;
        kk = maximo(k,kk);

   for(i=1;i<=kk;i++)
        {// calentamiento
            unsigned long zz;
            zz = girar_dec1();
        }
  }

 /////////////////////////////////////////////////////////////////////////////////////
 unsigned long girar_dec1(void)
        {// otra relacion de decimacion mas alta 0.25
            static int k=7;

     unsigned long z;
     unsigned long x[4];

            int i,j,kk,veces;

            (x[1]=a1,x[2]=a2,x[3]=a3,x[4]=a4);
```

```
                kk = 0;
                for(i=1;i<=4;i++)
                {
                        j = x[i]&15;
                        kk = kk + j;
                }

                k = (k+kk)%16;
        veces =1;
                if(k>=12) veces = 2; //decimacion 12/16=0.75     4/16=0.25

//              printf(" veces = %i  \n",veces);
        for(i=1; i<=veces;i++)
                {
                a1 = ~(a1^(a1<<5) + a2 + 1234567);
                a2 = a2^(a2>>13) + a1 + 76543;


                a3 = a3^(a3<<6)+ a4 + 314151;
                a4 = ~(a4^(a4>>7)+ a3 + 349765);

                z = (a1^a3) + (a2^a4);
        }

        return (z);
          }


//////////////////////////////////////////////////////////////////////////////////////


  void init_girar_dec2(unsigned long x1,unsigned long x2,unsigned long x3,unsigned long
x4)

  {
    int k=15;
    int kk;
         int i;
//    int maximo();
         unsigned long girar_dec2();

    (b1=x1,b2=x2,b3=x3,b4=x4);

         kk = (b1+b2+b3+b4)&63;
```

```
            kk = maximo(k,kk);

      for(i=1;i<=kk;i++)
            {// calentamiento
                  unsigned long zz;
                  zz = girar_dec2();
            }
   }
////////////////////////////////////////////////////////////////////////////////

  unsigned long girar_dec2(void)
            {// otra relacion de decimacion mas alta 0.25
                  static int k=13;

         unsigned long z;
         unsigned long x[4];

                  int i,j,kk,veces;

                  (x[1]=b1,x[2]=b2,x[3]=b3,x[4]=b4);

                  kk = 0;
                  for(i=1;i<=4;i++)
                  {
                        j = x[i]&15;
                        kk = kk + j;
                  }

                  k = (k+kk)%16;
                  veces =1;
                  if(k>=12) veces = 2; //decimacion 12/16=0.75     4/16=0.25

//                printf(" veces = %i  \n",veces);
         for(i=1; i<=veces;i++)
                  {
                    b1 = ~(b1^(b1<<5) + b2 + 7654321);
                    b2 = b2^(b2>>13) + b1 + 1234567;

                    b3 = b3^(b3<<6)+ b4 + 151413;
                   b4 = ~(b4^(b4>>7)+ b3 + 567943);

                   z = (b1^b3) + (b2^b4);

                  }
```

```
        return (z);
           }


//////////////////////////////////////////////////////////////////////////////////
     int short logico=1;

     double calcular_doble(unsigned long n1,unsigned long n2)
//                    unsigned long n1,n2;

           {
        unsigned long girar_dec1();
        unsigned long girar_dec2();

                    unsigned long n_up,n_low;

                    double inv_2ala32 = 0.00000000023283064365386962890625;
                    double u;
//printf("entrando a calcular_doble>>>> n1 ,   n2 = %u  %u \n",n1,n2);
       if(logico)
                   {
                           n_up = n1>>5;
                           n_low = n2>>6;
                   }
                   else
                   {
                           n_up = n2>>5;
                           n_low = n1>>6;
                   }

                   u = (n_up*67108864 + n_low)*inv_2ala32;
// colocar cambio aleatorio
                   logico = !logico;
//printf(" calcular_doble =  %f \n",u);
                   return u;
           }
//////////////////////////////////////////////////////////////////////////////////
        double Tabla[8][8];

            void init_girar_me(unsigned long i1,unsigned long i2,unsigned long i3,unsigned
long i4,unsigned long j1,unsigned long j2,unsigned long j3,unsigned long j4)

                    {
```

```
                unsigned long girar_dec1();
                unsigned long girar_dec2();

                unsigned long n1,n2;

                int i,j;

        init_girar_dec1(i1,i2,i3,i4);
        init_girar_dec2(j1,j2,j3,j4);

                     for(i=0;i<=7;i++)
                     {

                           for(j=0;j<=7; j=j+2)
                           {
                                 n1 = girar_dec1();
//                           printf(" n1== >> %u \n", n1);

                         n2 = girar_dec2();
//                           printf(" n2== >> %u \n", n2);

                                 Tabla[i][j] = calcular_doble(n1,n2);
                         Tabla[i][j+1] = calcular_doble(n1,n2);
//                               printf(" ij= %f ij+1= %f \n", Tabla[i][j],Tabla[i][j+1]);
                           }
                     }

                }

////////////////////////////////////////////////////////////////////////////////////////

                double girar_me(void)

          {
                unsigned long girar_dec1();
                unsigned long girar_dec2();

                unsigned long n1,n2;

                int i1,i2;
                int i,j;

                double u,w;
```

```
//                    printf(" entrando a girar_me la tabla es \n");
                            for(i=0;i<=7;i++)
                            {

                                     for(j=0;j<=7; j=j+2)
                                     {

//                                            printf("  i=  %i  j=  %i  ij=  %f  ij+1=  %f  \n",i,j,
Tabla[i][j],Tabla[i][j+1]);
                                     }
                            }

                    n1 = girar_dec1();
                    n2 = girar_dec2();

                    i1 = n1%8;
                    i2 = n2%8;
//              printf(" girar_me i1== %i i2== %i \n", i1,i2);
                    u = calcular_doble(n1,n2);
//              printf(" girar_me u== %f \n", u);

                    w = Tabla[i1][i2];
//              printf(" girar_me w== %f \n", w);

        Tabla[i1][i2] = u;
//              printf(" girar_me Tabla== %f \n",Tabla[i1][i2] );

                    return w;

            }

///////////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////////

                   int s=0;

                   unsigned long reiniciar_girar_me(unsigned long w)
                   {// reinicar girar_me_me
                     unsigned long a_o_v,b_o_v,c_o_v,d_o_v;

                     unsigned long a_b_v,b_b_v,c_b_v,d_b_v;

//                   int limite=200;
```

```
                    int limite= 400000;

                    s = s + (w>>12)%10;
//                     printf(" w== %u  (w>12)== %u \n", w, (w>12)%10);
//                     printf(" s== %d \n",s);

                    if(s>limite)
                                        {
// inicializa
                              printf(" *** entrando reiniciar_girar_me *** \n");
//                    printf(" se paso s=  %d \n",s);
                  s= (w>>12)%10;
                    printf(" nuevo inicial s=  %d \n",s);
// guarda actuales
                          a_o_v=  a1;
                          b_o_v=  a2;
                          c_o_v=  a3;
                          d_o_v=  a4;

                          a_b_v=  b1;
                          b_b_v=  b2;
                          c_b_v=  b3;
                          d_b_v=  b4;
// cambia estado uno por el otro
                          a1  = a_b_v;
                          a2  = b_b_v;
                          a3  = c_b_v;
                          a4  = d_b_v;

                          b1  = a_o_v;
                          b2  = b_o_v;
                          b3  = c_o_v;
                          b4  = d_o_v;

                                   printf("a1==> %u \n",a_o_v);
                                   printf("b1==> %u \n",a_o_v);
                                        }
                    return s;
          }

/////////////////////////////////////////////////////////////////////////////


                    int vez_dual=1;
```

```
    unsigned long girar_me_me(void)
{
        unsigned long y3;
    double yy;

        {
                if(vez_dual==1)
                {
        init_girar_me(ii1,ii2,ii3,ii4, jj1, jj2, jj3, jj4);
                        vez_dual =2;

        yy = girar_me();
//      printf(" vez 1 yy== %f \n", yy);

                        y3 = 4294967296*yy; // conversion a UL
                        return y3;
                }
      else
                {
        yy = girar_me();
//      printf(" vez 2 yy== %f \n", yy);

                        y3 = 4294967296*yy;

                        reiniciar_girar_me(y3);

        return y3;
                }
        }
 }
/////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

//  programa de prueba
        unsigned long main(void)
{
    int i;
    unsigned long z;

        printf("        prueba girar_me_me \n");
        for(i=1;i<=10;i++)
        {
```

```
            z = girar_me_me();

            printf("iter i-> %d numero en main ==> %u \n\n",i, z);
        }

        printf("\n          en 10 iteraciones  x10 es 2445914736  \n");
        printf("iter =  %d el numero en main es ==>   %u \n",i-1,z);

    return (z);
}
```

////////////////////////////////////////////////////////////////////////////////
Resultados Tuftest
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

```
Birthday spacings test: 4096 birthdays, 2^32 days in year
Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed    93   339   702  1011   943   823   524   295   160    67    43
(O-E)^2/E   0.0   2.0   1.3   1.2   1.2   2.2   0.0   0.0   0.8   0.0   0.1
            Birthday Spacings: Sum(O-E)^2/E=  8.935, p= 0.462

Euclid's algorithm:
 p-value, steps to gcd:   0.578012
 p-value, dist. of gcd's: 0.463816

 Gorilla test for 2^26 bits, positions 0 to 31:
 Note: lengthy test---for example, ~20 minutes for 850MHz PC
 Bits  0 to  7---> 0.076 0.912 0.552 0.457 0.717 0.264 0.348 0.872
 Bits  8 to 15---> 0.996 0.291 0.632 0.715 0.124 0.144 0.524 0.205
 Bits 16 to 23---> 0.492 0.679 0.850 0.041 0.127 0.324 0.870 0.990
 Bits 24 to 31---> 0.530 0.863 0.234 0.230 0.034 0.051 0.823 0.384
 KS test for the above 32 p values:  0.192
Press any key to continue
```

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

### 3.3.2.- resultados pruebas Tuftest a girar_me

Los valores resultantes de las pruebas con Tuftest con 100 muestras independientes aparecen en la tabla siguiente.

| | | tuftest | _girar_me | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **cumpleaños** | **gcd Euc** | **lides** | **gorila** | | | | | **rango** |
| 1 | **0.575** | **0.34697** | **0.37272** | **0.117** | **5** | **4** | **4** | **2** | **15** |
| 2 | **0.750** | **0.22589** | **0.86965** | **0.541** | **3** | **3** | **2** | **5** | **13** |
| 3 | **0.983** | **0.22468** | **0.19517** | **0.437** | **1** | **3** | **2** | **5** | **11** |
| 4 | **0.125** | **0.39933** | **0.18815** | **0.181** | **2** | **4** | **2** | **2** | **10** |
| 5 | **0.965** | **0.75340** | **0.90685** | **0.186** | **1** | **3** | **1** | **2** | **7** |
| 6 | **0.535** | **0.60618** | **0.45370** | **-7.102** | **5** | **4** | **5** | **-70** | **-1** |
| 7 | **0.954** | **0.39450** | **0.53717** | **0.599** | **1** | **4** | **5** | **5** | **15** |
| 8 | **0.901** | **0.28839** | **0.68425** | **-4.191** | **1** | **3** | **4** | **-40** | **-1** |
| 9 | **0.410** | **0.62813** | **0.24529** | **-5.788** | **5** | **4** | **3** | **-56** | **-1** |
| 10 | **0.721** | **0.35022** | **0.14920** | **0.310** | **3** | **4** | **2** | **4** | **13** |
| 11 | **0.306** | **0.41244** | **0.28672** | **0.249** | **4** | **5** | **3** | **3** | **15** |
| 12 | **0.244** | **0.43767** | **0.86098** | **0.377** | **3** | **5** | **2** | **4** | **14** |
| 13 | **0.511** | **0.73885** | **0.75391** | **-5.856** | **5** | **3** | **3** | **-57** | **-1** |
| 14 | **0.239** | **0.38102** | **0.74198** | **0.025** | **3** | **4** | **3** | **1** | **11** |
| 15 | **0.999** | **0.39777** | **0.27537** | **-5.064** | **1** | **4** | **3** | **-49** | **-1** |
| 16 | **0.934** | **0.17955** | **0.54951** | **0.602** | **1** | **2** | **5** | **4** | **12** |
| 17 | **0.526** | **0.25705** | **0.35044** | **0.589** | **5** | **3** | **4** | **5** | **17** |
| 18 | **0.725** | **0.67222** | **0.40029** | **0.341** | **3** | **4** | **5** | **4** | **16** |
| 19 | **0.253** | **0.05305** | **0.76382** | **0.265** | **3** | **1** | **3** | **3** | **10** |
| 20 | **0.777** | **0.82520** | **0.59313** | **-6.885** | **3** | **2** | **5** | **-67** | **-1** |
| 21 | **0.191** | **0.17403** | **0.83621** | **0.411** | **2** | **2** | **2** | **5** | **11** |
| 22 | **0.174** | **0.74774** | **0.21114** | **-4.869** | **2** | **3** | **3** | **-47** | **-1** |
| 23 | **0.017** | **0.27226** | **0.44764** | **-5.831** | **1** | **3** | **5** | **-57** | **-1** |
| 24 | **0.406** | **0.06379** | **0.63227** | **0.303** | **5** | **1** | **4** | **4** | **14** |
| 25 | **0.301** | **0.86586** | **0.51640** | **0.596** | **4** | **2** | **5** | **5** | **16** |
| 26 | **0.017** | **0.56909** | **0.39493** | **0.143** | **1** | **5** | **4** | **2** | **12** |
| 27 | **0.904** | **0.61601** | **0.66048** | **-4.921** | **1** | **4** | **4** | **-48** | **-1** |
| 28 | **0.733** | **0.65406** | **0.22544** | **0.415** | **3** | **4** | **3** | **5** | **15** |
| 29 | **0.101** | **0.23147** | **0.14405** | **0.496** | **2** | **3** | **2** | **5** | **12** |
| 30 | **0.835** | **0.33131** | **0.11252** | **0.099** | **2** | **4** | **2** | **1** | **9** |
| 31 | **0.625** | **0.41279** | **0.67924** | **0.354** | **4** | **5** | **4** | **4** | **17** |
| 32 | **0.274** | **0.77076** | **0.70410** | **-4.657** | **3** | **3** | **3** | **-45** | **-1** |
| 33 | **0.938** | **0.60429** | **0.44692** | **0.135** | **1** | **4** | **5** | **2** | **12** |
| 34 | **0.806** | **0.43092** | **0.36096** | **0.086** | **2** | **5** | **4** | **1** | **12** |
| 35 | **0.829** | **0.14235** | **0.53378** | **0.352** | **2** | **2** | **5** | **4** | **13** |
| 36 | **0.965** | **0.59107** | **0.26449** | **0.516** | **1** | **5** | **3** | **5** | **14** |
| 37 | **0.720** | **0.41208** | **0.91548** | **0.146** | **3** | **5** | **1** | **2** | **11** |
| 38 | **0.876** | **0.35532** | **0.64040** | **0.428** | **2** | **4** | **4** | **5** | **15** |
| 39 | **0.236** | **0.80859** | **0.36533** | **-4.845** | **3** | **2** | **4** | **-47** | **-1** |
| 40 | **0.410** | **0.60356** | **0.57560** | **0.224** | **5** | **4** | **5** | **3** | **17** |
| 41 | **0.375** | **0.62537** | **0.36572** | **-4.952** | **4** | **4** | **4** | **-48** | **-1** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 42 | **0.573** | **0.79525** | **0.82735** | **0.336** | **5** | **3** | **2** | **4** | **14** |
| 43 | **0.830** | **0.62599** | **0.07216** | **0.192** | **2** | **4** | **1** | **2** | **9** |
| 44 | **0.731** | **0.30828** | **0.38864** | **0.371** | **3** | **4** | **4** | **4** | **15** |
| 45 | **0.832** | **0.80507** | **0.53857** | **0.140** | **2** | **2** | **5** | **2** | **11** |
| 46 | **0.659** | **0.59381** | **0.73778** | **-5.663** | **4** | **5** | **3** | **-55** | **-1** |
| 47 | **0.392** | **0.53561** | **0.58318** | **0.591** | **4** | **5** | **5** | **5** | **19** |
| 48 | **0.403** | **0.77686** | **0.89455** | **-6.218** | **5** | **3** | **2** | **-61** | **-1** |
| 49 | **0.665** | **0.81637** | **0.41944** | **0.549** | **4** | **2** | **5** | **5** | **16** |
| 50 | **0.013** | **0.27109** | **0.91416** | **0.558** | **1** | **3** | **1** | **5** | **10** |
| 51 | **0.323** | **0.58900** | **0.49931** | **0.593** | **4** | **5** | **5** | **5** | **19** |
| 52 | **0.033** | **0.44923** | **0.88766** | **-5.192** | **1** | **5** | **2** | **-50** | **-1** |
| 53 | **0.664** | **0.77115** | **0.32043** | **0.636** | **4** | **3** | **4** | **4** | **15** |
| 54 | **0.743** | **0.58013** | **0.37241** | **0.112** | **3** | **5** | **4** | **2** | **14** |
| 55 | **0.894** | **0.19029** | **0.75996** | **0.385** | **2** | **2** | **3** | **4** | **11** |
| 56 | **0.540** | **0.40413** | **0.84862** | **-4.267** | **5** | **5** | **2** | **-41** | **-1** |
| 57 | **0.237** | **0.76825** | **0.59490** | **0.168** | **3** | **3** | **5** | **2** | **13** |
| 58 | **0.696** | **0.59930** | **0.65370** | **0.052** | **4** | **5** | **4** | **1** | **14** |
| 59 | **0.722** | **0.84234** | **0.82580** | **-6.328** | **3** | **2** | **2** | **-62** | **-1** |
| 60 | **0.879** | **0.53189** | **0.20633** | **0.115** | **2** | **5** | **3** | **2** | **12** |
| 61 | **0.835** | **0.77874** | **0.96041** | **0.123** | **2** | **3** | **1** | **2** | **8** |
| 62 | **0.536** | **0.25832** | **0.67919** | **-4.508** | **5** | **3** | **4** | **-44** | **-1** |
| 63 | **0.857** | **0.25395** | **0.36810** | **0.252** | **2** | **3** | **4** | **3** | **12** |
| 64 | **0.035** | **0.16028** | **0.22338** | **0.212** | **1** | **2** | **3** | **3** | **9** |
| 65 | **0.740** | **0.21544** | **0.74551** | **0.063** | **3** | **3** | **3** | **1** | **10** |
| 66 | **0.547** | **0.62130** | **0.79476** | **-5.519** | **5** | **4** | **3** | **-54** | **-1** |
| 67 | **0.080** | **0.42450** | **0.68263** | **-5.726** | **1** | **5** | **4** | **-56** | **-1** |
| 68 | **0.956** | **0.34820** | **0.18568** | **-4.685** | **1** | **4** | **2** | **-45** | **-1** |
| 69 | **0.136** | **0.24489** | **0.22841** | **0.281** | **2** | **3** | **3** | **3** | **11** |
| 70 | **0.483** | **0.84870** | **0.84638** | **0.221** | **5** | **2** | **2** | **3** | **12** |
| 71 | **0.287** | **0.17784** | **0.44758** | **-4.141** | **3** | **2** | **5** | **-40** | **-1** |
| 72 | **0.377** | **0.18503** | **0.93212** | **0.401** | **4** | **2** | **1** | **5** | **12** |
| 73 | **0.010** | **0.20089** | **0.70010** | **0.006** | **1** | **3** | **3** | **1** | **8** |
| 74 | **0.685** | **0.61260** | **0.12965** | **0.587** | **4** | **4** | **2** | **5** | **15** |
| 75 | **0.765** | **0.61659** | **0.31381** | **0.493** | **3** | **4** | **4** | **5** | **16** |
| 76 | **0.386** | **0.30343** | **0.49672** | **-4.117** | **4** | **4** | **5** | **-40** | **-1** |
| 77 | **0.484** | **0.91886** | **0.10145** | **0.355** | **5** | **1** | **2** | **4** | **12** |
| 78 | **0.772** | **0.92576** | **0.54554** | **-6.365** | **3** | **1** | **5** | **-62** | **-1** |
| 79 | **0.306** | **0.79745** | **0.57286** | **-4.808** | **4** | **3** | **5** | **-47** | **-1** |
| 80 | **0.533** | **0.72171** | **0.44892** | **0.244** | **5** | **3** | **5** | **3** | **16** |
| 81 | **0.248** | **0.48438** | **0.77677** | **0.448** | **3** | **5** | **3** | **5** | **16** |
| 82 | **0.070** | **0.24709** | **0.77090** | **0.278** | **1** | **3** | **3** | **3** | **10** |
| 83 | **0.591** | **0.55762** | **0.80554** | **0.104** | **5** | **5** | **2** | **2** | **14** |
| 84 | **0.639** | **0.81298** | **0.57284** | **-4.483** | **4** | **2** | **5** | **-43** | **-1** |
| 85 | **0.972** | **0.25923** | **0.42242** | **-7.987** | **1** | **3** | **5** | **-78** | **-1** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 86 | 0.010 | 0.09068 | 0.46838 | 0.288 | 1 | 1 | 5 | 3 | 10 |
| 87 | 0.850 | 0.54517 | 0.90977 | 0.304 | 2 | 5 | 1 | 4 | 12 |
| 88 | 0.165 | 0.72107 | 0.74604 | 0.558 | 2 | 3 | 3 | 5 | 13 |
| 89 | 0.369 | 0.22962 | 0.46515 | -4.226 | 4 | 3 | 5 | -41 | -1 |
| 90 | 0.855 | 0.76872 | 0.62701 | 0.152 | 2 | 3 | 4 | 2 | 11 |
| 91 | 0.251 | 0.91418 | 0.35234 | 0.127 | 3 | 1 | 4 | 2 | 10 |
| 92 | 0.790 | 0.27054 | 0.14559 | 0.551 | 3 | 3 | 2 | 5 | 13 |
| 93 | 0.555 | 0.73407 | 0.64186 | -4.276 | 5 | 3 | 4 | -41 | -1 |
| 94 | 0.111 | 0.90649 | 0.66158 | 0.027 | 2 | 1 | 4 | 1 | 8 |
| 95 | 0.121 | 0.62047 | 0.62304 | 0.472 | 2 | 4 | 4 | 5 | 15 |
| 96 | 0.636 | 0.22050 | 0.41168 | -4.648 | 4 | 3 | 5 | -45 | -1 |
| 97 | 0.998 | 0.53907 | 0.47191 | -8.265 | 1 | 5 | 5 | -81 | -1 |
| 98 | 0.412 | 0.50044 | 0.45254 | -4.436 | 5 | 5 | 5 | -43 | -1 |
| 99 | 0.227 | 0.44991 | 0.45995 | 0.092 | 3 | 5 | 5 | 1 | 14 |
| 100 | 0.298 | 0.60256 | 0.85748 | 0.380 | 3 | 4 | 2 | 4 | 13 |
| | ===> final | tuftest < | === | | | | | | |

### 3.4.- dinámico_CR

#### 3.4.1.- código dinámico_CR

El programa en C sigue a continuación.

```
#include <stdio.h>
//  dinámico de [1] Gonzalez y otros GNSA33

/* Este archivo contiene un simple main() para implantar dinámico_CR
   de dinámico que usa 2 instancias de dinámico que es la version
   en entero sin signo de 32 bits de las ecuaciones 16, 17 y 18, en
   doble precision que transforman un sistema dinámico en uno simetrico.

   La ecuaciones basicas de dinámico son:

       xn+1 = [(a1+b1yn + c1zn)xn + d1yn + e1zn] (mod 1)    (16)

       yn+1 = [(a2+b2zn + c2xn)yn + d2zn + e2xn] (mod 1)    (17)

       zn+1 = [(a3+b3xn + c3yn)zn + d3xn + e3yn] (mod 1)    (18)

   tomadas de [1] Gonzalez, Moreno y Guerrero, Non-invertible Transformations
   and Spaciotemporal Randomness. 2005.
   Los valores a1,a2,a3,b1,b2,b3,c1,c2,c3,d1,d2,d3,e1,e2,y e3 son constantes reales
   mayores que 1.
```

xn, yyn, y zn en dinámico1 y xxn, yyyn, yzzn en dinámico2 son los valores iniciales del algoritmo dinámico_CR.

Se toman 2 instancias distintas dinámico1 y dinámico2. En base a estos dos valores generados se construye como la suma el valor generado.

En una tabla de mezcla se almacenan los valores generados por dinámico, tabla de la cual se extrae al azar el valor entero generado por dinámico_CR.

La organizacion de dinámico_CR es:

```
dinámico_CR
     |   // inicialización
     |---| iniciar_dinámico_CR
     |   | // calentar dinámico_g1 y dinámico_g2 y llenar Tabla de mezcla
         |   |--- dinámico_g1
         |   |--- dinámico_g2
         |   |--- dinámico
     |
         |   // generar valores
         |---| generar_dinámico_CR
         |   |   // extraer al azar Tabla(i), reemplazar valor
         |   |--- dinámico_g1
         |   |--- dinámico_g2
         |   |--- dinámico
         |        |
                   |        |--- dinámico1
                   |        |--- dinámico_g1
                   |        |--- dinámico2
                   |        |--- dinámico_g2
                   |
         |   // reinicilizacion aleatoria de dinámico_CR
     |--- reiniciar_dinámico_CR


  programador R.A. Pastoriza       sep/2007- mayo/2008
*/
/////////////////////////////////////////////////////////////////////////////
// valores iniciales de dinámico_CR
// valores iniciales de las variables de estado dinámico1
        static double xn= 0.1, yyn= 0.1, zn= 0.1;

// valores iniciales de las variables de estado dinámico2
```

```
        static double xxn= 0.1, yyyn= 0.1, zzn= 0.1;
///////////////////////////////////////////////////////////////////////////

// constantes de dinámico1, las constantes originales de [1] Gonzaleza y otros.
   double a1 =1.3, b1=3.14159265358979323, c1=2.6 , d1= 1.5, e1= 1.1;
   double a2 =4.6, b2=2.1, c2=1.1 , d2= 2.2, e2= 7.1;
   double a3 =2.9, b3=5.4, c3=8.7 , d3= 4.5, e3= 1.9;


///////////////////////////////////////////////////////////////////////////
   double dinámico_g1(void)
        {
   double hh;

    xn = ((a1 + b1*yyn + c1*zn)*xn + d1*yyn + e1*zn);
    xn = xn - (int)xn;          // eq 16

    yyn = ((a2 + b2*zn + c2*xn)*yyn + d2*zn + e2*xn);
    yyn = yyn - (int)yyn;       // eq 17

    zn = ((a3 + b3*xn + c3*yyn)*zn + d3*xn + e3*yyn);
    zn = zn -(int)zn;           // eq 18

    hh = (xn + yyn + zn);       // con la suma pasa bien
    hh = hh - (int)hh;

   return hh;
        }
///////////////////////////////////////////////////////////////////////////

        unsigned long dinámico1(void)
        { // llevar a UL el doble de dinámico1
                unsigned long ww;

                double dinámico_g1();

                ww = 4294967296*dinámico_g1();

                return ww;
        }
///////////////////////////////////////////////////////////////////////////

// constantes de dinámico2, distintas de las iniciales de [1]] Gonzalez y otros
```

```
        double aa1 =2.11345, bb1=1.141592653, cc1=1.776666 , dd1= 2.5789123, ee1=
4.98712345;
       double aa2 =3.6987065, bb2=0.312435, cc2=2.19998765 , dd2= 2.2468, ee2=
5.1234567;
   double aa3 =5.909876503, bb3=3.455667788, cc3=1.789789789 , dd3= 3.5654321, ee3=
3.901234567;

//////////////////////////////////////////////////////////////////////////////
   double dinámico_g2(void)
        {
   double hh;

    xxn = ((aa1 + bb1*yyyn + cc1*zzn)*xxn + dd1*yyyn + ee1*zzn);
    xxn = xxn - (int)xxn;          // eq 16

    yyyn = ((aa2 + bb2*zzn + cc2*xxn)*yyyn + dd2*zzn + ee2*xxn);
    yyyn = yyyn - (int)yyyn;       // eq 17

    zzn = ((aa3 + bb3*xxn + cc3*yyyn)*zzn + dd3*xxn + ee3*yyyn);
    zzn = zn -(int)zn;             // eq 18

    hh = (xxn + yyyn + zzn);       // con la suma pasa bien
    hh = hh - (int)hh;

    return hh;
        }
//////////////////////////////////////////////////////////////////////////////

        unsigned long dinámico2(void)
        {// a entero largo sin signo
                unsigned long ww;

                double dinámico_g2();

                ww = 4294967296*dinámico_g2();

                return ww;
        }
//////////////////////////////////////////////////////////////////////////////

   unsigned long dinámico(void)
        {// componer resultado
      unsigned long w;
```

```
                    unsigned long dinámico1();
                    unsigned long dinámico2();

                    w = (dinámico1() + dinámico2());

      return w;
        }

//////////////////////////////////////////////////////////////////////////////
/////////////////////////////////////////////////////////////////////////////

    unsigned long Tabla[128];

    void iniciar_dinámico_CR(void)

      { // iniciar dinámico_CR
              double dinámico_g1();
              double dinámico_g2() ;

    int k= 39;
              int kk1,kk2;
         int i,j;

// llenar tabla de mezcla con 0

              for(i=0;i<=127;i++)
              {
                Tabla[i] = 0;
              }


//    calentamiento de dinámico, diferente por cada componente

//                printf(" iniciar calentamiento \n");

         kk1 = 100*(a1+b1+c1+d1+e1)- (int) 100*(a1+b1+c1+d1+e1);
         kk2 = 100*(aa1+bb1+cc1+dd1+ee1)- (int) 100*(aa1+bb1+cc1+dd1+ee1);

         kk1 = kk1&63;
         kk2 = kk2&63;;         // tamanio variable del calentamiento


//      printf(" kk1 == %d kk2 == %d\n",kk1,kk2);
```

```
            for(i=1;i<=kk1;i++)
                {// calentamiento dinámicog1
                        dinámico_g1();
                }

                for(j=1;j<=kk2;j++)
                {// calentamiento dinámicog1
                        dinámico_g2();
                }


  // llenar la tabla inicial

                for(i=0;i<=127;i++)
                {
        Tabla[i] = dinámico();
                }

           }

////////////////////////////////////////////////////////////////////////////
   int kswitch = 1;
         unsigned long generar_dinámico_CR(unsigned long z)
         {
                int i1;

                double dinámico_g1();
                double dinámico_g2();

// generar valores
                if(kswitch ==1)
                {// alternar calculo del indice
                   i1 = (int)128*dinámico_g1();   // indice de extraccion
//                                printf(" i1== %d \n", i1);
       }
                else
                {
                        i1 = (int)128*dinámico_g2();   // indice de extraccion
//                                                printf(" i1== %d \n", i1);
                   kswitch = 0;
                }

                z = Tabla[i1]; // valor extraido de la Tabla de mezcla
```

```
// reemplazar valor extraido

                Tabla[i1] = dinámico();

//              printf(" generar_dinámico_CR z== %f \n",z);

                return z;
        }

////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////
                int s=0;

            void reiniciar_dinámico_CR(unsigned long z)
                {
                 double x1_o, y1_o, z1_o;
             double x2_o, y2_o, z2_o;

//               int limite=4;
                 int limite= 600000;

                 s = s + (z)%10;
//                  printf(" z== %u  (z)%10== %d \n", z, (z)%10);
//                  printf(" s== %d \n",s);

                 if(s>limite)
                                        {
// inicializa
//                printf(" *** entrando reiniciar_porfiado_CR *** \n");
//                printf(" se paso s=  %d \n",s);
                s= (z)%10;
//                 printf(" nuevo inicial s=  %d \n",s);

// guarda estado
                x1_o = xn;
                                        y1_o = yyn;
                                        z1_o = zn;
//                                     printf("xn original %f zzn original %f\n",xn,zzn);
                x2_o = xxn;
                                        y2_o = yyyn;
                                        z2_o = zzn;

// cambia estado
```

```
                    xn  = z2_o;
                                        yyn = x2_o;
                                        zn  = y2_o;

                                        xxn  = z1_o;
                                        yyyn = x1_o;
                                        zzn  = y1_o;
//                                      printf("xn cambiado==> %f \n",xn);

                                        }

            }

//////////////////////////////////////////////////////////////////////////////

            int vez_dinámico =1;

      unsigned long dinámico_CR()
{
      unsigned long yy;
      unsigned long z,zz;

            void iniciar_dinámico_CR();
//          unsigned long generar_dinámico_CR();

            {
                  if(vez_dinámico==1)
                  {// inicializacion del generador
//                      printf(" entrando iniciar_dinámico %d \n");

                        iniciar_dinámico_CR();

                        yy = generar_dinámico_CR(z);
 //               printf(" entero en generar_dinámico_CR %u \n", yy);

                        vez_dinámico =2;

                        return yy;
                  }
         else
                  {// generacion

                        yy = generar_dinámico_CR(z);
//              printf(" entero en generar_dinámico_CR %u \n", yy);
```

```
            reiniciar_dinámico_CR(z);

//                                  printf(" estado s en reiniciar_dinámico_CR ==>    %1u
\n",s);

                            return yy;
                    }
            }
 }


/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////

// programa de prueba
            unsigned long main(void)
{
     int i;
     unsigned long z;

            printf("        prueba dinámico_CR \n");
//          void iniciar_dinámico_CR();
            unsigned long dinámico();

            for(i=1;i<=10;i++)
            {
                    z = dinámico_CR();
               printf("iter i-> %d numero en main ==> %u \n\n",i, z);

            }

            printf("\n          en 10 iteraciones  x10 es 381488454   \n");
            printf("iter =  %d el numero en main es ==>   %u \n",i-1,z);

     return (z);
}

/////////////////////////////////////////////////////////////////////////////
```

Resultados Tuftest

+++++++++++++++++++++++++++++++++++++++++

Birthday spacings test: 4096 birthdays, 2^32 days in year

Table of Expected vs. Observed counts:

| Duplicates | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | >=10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Expected | 91.6 | 366.3 | 732.6 | 976.8 | 976.8 | 781.5 | 521.0 | 297.7 | 148.9 | 66.2 | 40.7 |
| Observed | 85 | 364 | 750 | 921 | 951 | 805 | 540 | 310 | 178 | 58 | 38 |
| (O-E)^2/E | 0.5 | 0.0 | 0.4 | 3.2 | 0.7 | 0.7 | 0.7 | 0.5 | 5.7 | 1.0 | 0.2 |

Birthday Spacings: Sum(O-E)^2/E= 13.573, p= 0.807

Euclid's algorithm:
p-value, steps to gcd:   0.501481
p-value, dist. of gcd's: 0.651473

Gorilla test for 2^26 bits, positions 0 to 31:
Note: lengthy test---for example, ~20 minutes for 850MHz PC
Bits  0 to  7---> 0.790 0.684 0.574 0.799 0.620 0.773 0.641 0.310
Bits  8 to 15---> 0.700 0.681 0.389 0.209 0.781 0.518 0.796 0.043
Bits 16 to 23---> 0.095 0.996 0.699 0.792 0.663 0.716 0.230 0.042
Bits 24 to 31---> 0.858 0.417 0.387 0.132 0.338 0.273 0.833 0.005
KS test for the above 32 p values:  0.523
Press any key to continue
+++++++++++++++++++++++++++++++++++++++++++

### 3.4.2.- resultados pruebas Tuftest a dinámico_CR

Los valores resultantes de las pruebas con Tuftest con 65 muestras independientes aparecen en la tabla siguiente.

| | | tuftest_ | dinámico_CR | |
|---|---|---|---|---|
| | **cumpleaños** | **gcd Eucl** | **ides** | **gorila** |
| 1 | **0.807000** | **0.501481** | **0.651473** | **0.523000** |
| 2 | **0.709000** | **0.786553** | **0.641553** | **-5.679000** |
| 3 | **0.655000** | **0.700388** | **0.850327** | **0.227000** |
| 4 | **0.125000** | **0.475103** | **0.418368** | **0.230000** |
| 5 | **0.425000** | **0.694599** | **0.419675** | **0.244000** |
| 6 | **0.876000** | **0.793951** | **0.512132** | **0.603000** |
| 7 | **0.327000** | **0.572095** | **0.263477** | **0.358000** |
| 8 | **0.427000** | **0.723019** | **0.579260** | **0.377000** |
| 9 | **0.064000** | **0.340000** | **0.591628** | **-4.321000** |
| 10 | **0.700000** | **0.915969** | **0.105014** | **-5.166000** |
| 11 | **0.932000** | **0.158693** | **0.927576** | **0.362000** |
| 12 | **0.120000** | **0.470904** | **0.207167** | **0.154000** |
| 13 | **0.040000** | **0.351764** | **0.825254** | **0.450000** |
| 14 | **0.731000** | **0.290112** | **0.361437** | **0.541000** |
| 15 | **0.929000** | **0.415313** | **0.691268** | **0.326000** |
| 16 | **0.867000** | **0.529880** | **0.394109** | **0.550000** |
| 17 | **0.324000** | **0.507726** | **0.764918** | **0.510000** |
| 18 | **0.927000** | **0.778561** | **0.336293** | **-4.371000** |
| 19 | **0.728000** | **0.465074** | **0.391674** | **0.252000** |
| 20 | **0.086000** | **0.405937** | **0.360636** | **0.529000** |
| 21 | **0.578000** | **0.432751** | **0.539869** | **0.197000** |
| 22 | **0.888000** | **0.506409** | **0.253004** | **-5.692000** |
| 23 | **0.961000** | **0.763494** | **0.600778** | **0.040000** |
| 24 | **0.093000** | **0.880844** | **0.440158** | **0.588000** |
| 25 | **0.551000** | **0.232156** | **0.473594** | **0.009000** |
| 26 | **0.561000** | **0.283279** | **0.941664** | **-4.771000** |
| 27 | **0.230000** | **0.280690** | **0.265374** | **0.019000** |
| 28 | **0.720000** | **0.345419** | **0.684582** | **-7.441000** |
| 29 | **0.401000** | **0.303084** | **0.428740** | **0.028000** |
| 30 | **0.926000** | **0.479086** | **0.163381** | **0.386000** |
| 31 | **0.549000** | **0.456697** | **0.655418** | **-5.067000** |
| 32 | **0.536000** | **0.197010** | **0.722113** | **0.284000** |
| 33 | **0.598000** | **0.182404** | **0.361427** | **0.063000** |
| 34 | **0.425000** | **0.290679** | **0.426575** | **-4.906000** |
| 35 | **0.648000** | **0.471319** | **0.503268** | **0.368000** |
| 36 | **0.392000** | **0.778438** | **0.756014** | **0.485000** |
| 37 | **0.110000** | **0.421218** | **0.199796** | **-4.253000** |
| 38 | **0.137000** | **0.535031** | **0.523318** | **-4.129000** |
| 39 | **0.834000** | **0.578366** | **0.778207** | **0.163000** |
| 40 | **0.420000** | **0.818843** | **0.714933** | **0.367000** |

| | | | | |
|---|---|---|---|---|
| 41 | **0.797000** | **0.716038** | **0.875604** | **0.266000** |
| 42 | **0.515000** | **0.760204** | **0.467465** | **0.342000** |
| 43 | **0.506000** | **0.412735** | **0.812864** | **0.060000** |
| 44 | **0.928000** | **0.493194** | **0.887645** | **-5.595000** |
| 45 | **0.061000** | **0.236290** | **0.308246** | **-4.128000** |
| 46 | **0.863000** | **0.792231** | **0.565478** | **0.129000** |
| 47 | **0.643000** | **0.241557** | **0.374696** | **0.557000** |
| 48 | **0.913000** | **0.508373** | **0.718133** | **0.502000** |
| 49 | **0.605000** | **0.773758** | **0.622661** | **0.182000** |
| 50 | **0.221000** | **0.340311** | **0.439808** | **-4.365000** |
| 51 | **0.358000** | **0.859637** | **0.075706** | **-6.806000** |
| 52 | **0.010000** | **0.623833** | **0.559331** | **-4.148000** |
| 53 | **0.339000** | **0.655611** | **0.954573** | **0.465000** |
| 54 | **0.127000** | **0.248579** | **0.501040** | **0.142000** |
| 55 | **0.506000** | **0.697068** | **0.346153** | **0.423000** |
| 56 | **0.038000** | **0.242444** | **0.837507** | **0.634000** |
| 57 | **0.652000** | **0.757826** | **0.649255** | **0.282000** |
| 58 | **0.670000** | **0.539440** | **0.491673** | **-4.781000** |
| 59 | **0.940000** | **0.919783** | **0.551589** | **0.296000** |
| 60 | **0.117000** | **0.372672** | **0.299687** | **0.211000** |
| 61 | **0.745000** | **0.367108** | **0.305826** | **-5.466000** |
| 62 | **0.740000** | **0.848274** | **0.215553** | **0.019000** |
| 63 | **0.804000** | **0.085543** | **0.225115** | **0.471000** |
| 64 | **0.308000** | **0.555069** | **0.255956** | **0.612000** |
| **65** | **0.564000** | **0.566335** | **0.861279** | **0.035000** |

## 3.5.- isaac_CR

### 3.5.1.- código isaac_CR

El programa en C sigue a continuación.

```
/*
isaac_CR es una version de isaac(readable.c) que devuelve un numero UL generado en
cada invocacion.

Adicionalmente tiene una reinicializacion por medio de las variables a, b, c , d, e,
f, g, h, de mix(a,b,c,d,e,f,g,h) rutina usada en randinit.

randinit es la rutina de [2] Jenkins para inicializar isaac, fija en la version original.

Se incorporaron las rutinas saca_una, que como su nombre lo indica sirve para
extraer un numero del vector randrsl[0..255], en orden secuencial 1, 2, ..., 255, 0
hasta agotar el vector y luego invoca nuevamente a isaac para rellenar el vector randrsl.

La inicializacion se hace vía randinit colocando valores externos para mix, como
variables globales y luego llamando a isaac para llenar el vector randrsl[0..255].

La organizacion de isaac_Cr es:

    isaac_CR
            | // inicializacion
            |--- randinit
            |       |--- mix
            |       |--- isaac
            |
            |// generacion
            |--- saca_uno
                 |--- isaac

referencia:

        http://burtleburtle.net/bob/rand/isaac.html
                        #IBAA
            ISAAC and RC4
            Robert J. Jenkins Jr., 1993-1996

programador R.A. Pastoriza   mayo/2008
*/
```

```
/*
------------------------------------------------------------------------------
readable.c: My random number generator, ISAAC.
(c) Bob Jenkins, March 1996, Public Domain
You may use this code in any way you wish, and it is free.  No warrantee.
* May 2008 -- made it not depend on standard.h
------------------------------------------------------------------------------
*/
#include <stdio.h>
#include <stddef.h>

/* a ub4 is an unsigned 4-byte quantity */

typedef  unsigned long int  ub4;

/* external results */
  ub4 randrsl[256], randcnt;

/* internal state */
static    ub4 mm[256];
static    ub4 aa=0, bb=0, cc=0;

// valores iniciales de isaac_CR
    ub4 a,b,c,d,e,f,g,h;
//    a=231;b=123;c=123;d=123;e=123;f=123;g=123;h=123;
    a=231;b=456;c=678;d=890;e=123;f=432;g=7777;h=2560;

void isaac()
{
   register ub4 i,x,y;

   cc = cc + 1;    /* cc just gets incremented once per 256 results */
   bb = bb + cc;   /* then combined with bb */

   for (i=0; i<256; ++i)
   {
     x = mm[i];
     switch (i%4)
     {
     case 0: aa = aa^(aa<<13); break;
     case 1: aa = aa^(aa>>6); break;
     case 2: aa = aa^(aa<<2); break;
     case 3: aa = aa^(aa>>16); break;
     }
```

```
    aa          = mm[(i+128)%256] + aa;
    mm[i]      = y  = mm[(x>>2)%256] + aa + bb;
    randrsl[i] = bb = mm[(y>>10)%256] + x;

    /* Note that bits 2..9 are chosen from x but 10..17 are chosen
       from y.  The only important thing here is that 2..9 and 10..17
       don't overlap.  2..9 and 10..17 were then chosen for speed in
       the optimized version (rand.c) */
    /* See http://burtleburtle.net/bob/rand/isaac.html
       for further explanations and analysis. */
  }
}


/* if (flag!=0), then use the contents of randrsl[] to initialize mm[]. */
#define mix(a,b,c,d,e,f,g,h) \
{ \
  a^=b<<11; d+=a; b+=c; \
  b^=c>>2;  e+=b; c+=d; \
  c^=d<<8;  f+=c; d+=e; \
  d^=e>>16; g+=d; e+=f; \
  e^=f<<10; h+=e; f+=g; \
  f^=g>>4;  a+=f; g+=h; \
  g^=h<<8;  b+=g; h+=a; \
  h^=a>>9;  c+=h; a+=b; \
}

void randinit(flag)
int flag;
{
  int i;
//   ub4 a,b,c,d,e,f,g,h;
  aa=bb=cc=0;
 //  a=b=c=d=e=f=g=h=0x9e3779b9;  /* the golden ratio */  //2 654 435 769 ????
                              // es la inicializacion original
  for (i=0; i<4; ++i)          /* scramble it */
  {
    mix(a,b,c,d,e,f,g,h);
  }

  for (i=0; i<256; i+=8)   /* fill in mm[] with messy stuff */
  {
    if (flag)                  /* use all the information in the seed */
    {
```

```
      a+=randrsl[i  ]; b+=randrsl[i+1]; c+=randrsl[i+2]; d+=randrsl[i+3];
      e+=randrsl[i+4]; f+=randrsl[i+5]; g+=randrsl[i+6]; h+=randrsl[i+7];
    }
    mix(a,b,c,d,e,f,g,h);
    mm[i  ]=a; mm[i+1]=b; mm[i+2]=c; mm[i+3]=d;
    mm[i+4]=e; mm[i+5]=f; mm[i+6]=g; mm[i+7]=h;
  }

  if (flag)
  {       /* do a second pass to make all of the seed affect all of mm */
    for (i=0; i<256; i+=8)
    {
      a+=mm[i  ]; b+=mm[i+1]; c+=mm[i+2]; d+=mm[i+3];
      e+=mm[i+4]; f+=mm[i+5]; g+=mm[i+6]; h+=mm[i+7];
      mix(a,b,c,d,e,f,g,h);
      mm[i  ]=a; mm[i+1]=b; mm[i+2]=c; mm[i+3]=d;
      mm[i+4]=e; mm[i+5]=f; mm[i+6]=g; mm[i+7]=h;
    }
  }

  isaac();            /* fill in the first set of results */
  randcnt=256;        /* prepare to use the first set of results */

}

////////////////////////////////////////////////////////////////////////////////
/*
rutina unsigned long saca_uno() por R.A. Pastoriza para extraer un solo valor por vez
de los 256 generados y almacenados en randrsl[]  por isaac
*/

  int indice = 0;
        unsigned long saca_uno()
        {// extraer un solo valor del vector randrsl[0..255]
                unsigned long z;

                indice = (indice+1)%256;
                if(indice ==0)
                {// calcular nuevos valores de randsrl[] llamando isaac
//                      printf(" indice == %d \n",indice);
                        z = randrsl[0];
                   isaac();
                }
                else
```

```
                {
                        z = randrsl[indice];
                }

                return (z);
        }
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
/*
rutina unsigned long isaac_CR() por R.A. Pastoriza para extraer cada vez un solo
valor de los 256 generados y almacenados en randrsl[]  por isaac, e inicializar
isaac solo en la primera llamada
*/

        int vez_isaac =1;

    unsigned long isaac_CR()
{// isaac con inicializacion separada y extraccion uno a uno de los valores de randrsl[]
    unsigned long yy;

        {
                if(vez_isaac==1)
                {// inicializacion del generador
//                              printf(" entrando iniciar_isaac_CR %d \n");

                        randinit(1); // inicializacion con verdad siempre
                        vez_isaac =2;
                        yy = saca_uno();

                        return yy;
                }
        else
                {// generacion

                        yy = saca_uno();
//              printf(" entero en isaac_CR %u \n", yy);

                        return yy;
                }
        }
 }
```

```
////////////////////////////////////////////////////////////////////////////////

void main()
{// isaac de uno por vez
     unsigned long w;
            ub4 i,j;
     aa=bb=cc=(ub4)0;

  for (i=0; i<256; ++i) mm[i]=randrsl[i]=(ub4)0;  // en cero
            printf("          prueba isaac_CR \n");

  for (i=1; i<=10; ++i)
  {
            w = isaac_CR();
            printf("iter i-> %d numero en main ==> %u \n\n",i, w);

  }
            printf("\n            en 10 iteraciones  x10 es 2805424826    \n");
            printf("iter =  %d el numero en main es ==>   %u \n",i-1,w);

}

/////////////////////////////////////////////////////////////////////////////////
```

### 3.5.2.- resultados de Tuftest a isaac_CR

Los resultados numericos aparecen a continuación.

| | | tuftest | _isaac_CR | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | cumpleaños | gcd Euc | lides | gorila | | | | | rango |
| 1 | 0.571 | 0.881 | 0.386 | 0.012 | 5 | 2 | 4 | 1 | 12 |
| 2 | 0.418 | 0.361 | 0.256 | 0.228 | 5 | 4 | 3 | 3 | 15 |
| 3 | 0.022 | 0.442 | 0.781 | 0.234 | 1 | 5 | 3 | 3 | 12 |
| 4 | 0.563 | 0.864 | 0.223 | 0.128 | 5 | 2 | 3 | 2 | 12 |
| 5 | 0.966 | 0.634 | 0.533 | 0.451 | 1 | 4 | 5 | 5 | 15 |
| 6 | 0.496 | 0.226 | 0.49 | 0.405 | 5 | 3 | 5 | 5 | 18 |
| 7 | 0.426 | 0.381 | 0.566 | -4.45 | 5 | 4 | 5 | -43 | -1 |
| 8 | 0.904 | 0.627 | 0.688 | -5.08 | 1 | 4 | 4 | -49 | -1 |
| 9 | 0.964 | 0.502 | 0.843 | 0.203 | 1 | 5 | 2 | 3 | 11 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 10 | **0.516** | **0.277** | **0.616** | **-5.01** | **5** | **3** | **4** | **-49** | **-1** |
| 11 | **0.701** | **0.127** | **0.279** | **0.195** | **3** | **2** | **3** | **2** | **10** |
| 12 | **0.623** | **0.606** | **0.531** | **-4.99** | **4** | **4** | **5** | **-48** | **-1** |
| 13 | **0.99** | **0.289** | **0.222** | **0.157** | **1** | **3** | **3** | **2** | **9** |
| 14 | **0.249** | **0.93** | **0.412** | **0.177** | **3** | **1** | **5** | **2** | **11** |
| 15 | **0.031** | **0.219** | **0.228** | **0.527** | **1** | **3** | **3** | **5** | **12** |
| 16 | **0.174** | **0.033** | **0.694** | **-5.08** | **2** | **1** | **4** | **-49** | **-1** |
| 17 | **0.513** | **0.735** | **0.596** | **0.169** | **5** | **3** | **5** | **2** | **15** |
| 18 | **0.242** | **0.673** | **0.278** | **0.493** | **3** | **4** | **3** | **5** | **15** |
| 19 | **0.869** | **0.623** | **0.089** | **0.639** | **2** | **4** | **1** | **4** | **11** |
| 20 | **0.498** | **0.713** | **0.474** | **-5.33** | **5** | **3** | **5** | **-52** | **-1** |
| 21 | **0.242** | **0.162** | **0.33** | **-5.36** | **3** | **2** | **4** | **-52** | **-1** |
| 22 | **0.644** | **0.681** | **0.156** | **0.117** | **4** | **4** | **2** | **2** | **12** |
| 23 | **0.668** | **0.826** | **0.146** | **-5.59** | **4** | **2** | **2** | **-54** | **-1** |
| 24 | **0.127** | **0.201** | **0.448** | **0.291** | **2** | **3** | **5** | **3** | **13** |
| 25 | **0.78** | **0.767** | **0.254** | **0.04** | **3** | **3** | **3** | **1** | **10** |
| 26 | **0.927** | **0.628** | **0.801** | **-4.29** | **1** | **4** | **2** | **-41** | **-1** |
| 27 | **0.733** | **0.753** | **0.184** | **0.426** | **3** | **3** | **2** | **5** | **13** |
| 28 | **0.808** | **0.188** | **0.506** | **-4.56** | **2** | **2** | **5** | **-44** | **-1** |
| 29 | **0.214** | **0.357** | **0.616** | **0.581** | **3** | **4** | **4** | **5** | **16** |
| 30 | **0.646** | **0.286** | **0.397** | **0.17** | **4** | **3** | **4** | **2** | **13** |
| 31 | **0.817** | **0.961** | **0.733** | **0.509** | **2** | **1** | **3** | **5** | **11** |
| 32 | **0.293** | **0.376** | **0.581** | **0.402** | **3** | **4** | **5** | **5** | **17** |
| 33 | **0.609** | **0.767** | **0.279** | **-4.94** | **4** | **3** | **3** | **-48** | **-1** |
| 34 | **0.06** | **0.531** | **0.075** | **0** | **1** | **5** | **1** | **1** | **-1** |
| 35 | **0.027** | **0.681** | **0.562** | **-5.36** | **1** | **4** | **5** | **-52** | **-1** |
| 36 | **0.453** | **0.279** | **0.231** | **0.329** | **5** | **3** | **3** | **4** | **15** |
| 37 | **0.735** | **0.491** | **0.645** | **0.169** | **3** | **5** | **4** | **2** | **14** |
| 38 | **0.721** | **0.761** | **0.195** | **-5.22** | **3** | **3** | **2** | **-51** | **-1** |
| 39 | **0.256** | **0.245** | **0.25** | **0.333** | **3** | **3** | **3** | **4** | **13** |
| 40 | **0.462** | **0.272** | **0.905** | **-5.18** | **5** | **3** | **1** | **-50** | **-1** |
| 41 | **0.972** | **0.192** | **0.616** | **0.506** | **1** | **2** | **4** | **5** | **12** |
| 42 | **0.276** | **0.758** | **0.302** | **-5.26** | **3** | **3** | **4** | **-51** | **-1** |
| 43 | **0.997** | **0.289** | **0.419** | **0.331** | **1** | **3** | **5** | **4** | **-1** |
| 44 | **0.865** | **0.541** | **0.673** | **0.069** | **2** | **5** | **4** | **1** | **12** |
| 45 | **0.182** | **0.463** | **0.708** | **0.627** | **2** | **5** | **3** | **4** | **14** |
| 46 | **0.395** | **0.079** | **0.704** | **0.35** | **4** | **1** | **3** | **4** | **12** |
| 47 | **0.252** | **0.608** | **0.41** | **0.343** | **3** | **4** | **5** | **4** | **16** |
| 48 | **0.935** | **0.481** | **0.754** | **0.462** | **1** | **5** | **3** | **5** | **14** |
| 49 | **0.755** | **0.342** | **0.321** | **0.256** | **3** | **4** | **4** | **3** | **14** |
| 50 | **0.177** | **0.145** | **0.655** | **-6.55** | **2** | **2** | **4** | **-64** | **-1** |
| 51 | **0.179** | **0.543** | **0.073** | **0.466** | **2** | **5** | **1** | **5** | **13** |
| 52 | **0.411** | **0.537** | **0.821** | **-6.38** | **5** | **5** | **2** | **-62** | **-1** |
| 53 | **0.491** | **0.825** | **0.576** | **0.288** | **5** | **2** | **5** | **3** | **15** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 54 | 0.219 | 0.463 | 0.302 | 0.637 | 3 | 5 | 4 | 4 | 16 |
| 55 | 0.415 | 0.197 | 0.652 | 0.17 | 5 | 2 | 4 | 2 | 13 |
| 56 | 0.92 | 0.792 | 0.978 | 0.149 | 1 | 3 | 1 | 2 | 7 |
| 57 | 0.18 | 0.504 | 0.063 | 0.435 | 2 | 5 | 1 | 5 | 13 |
| 58 | 0.993 | 0.655 | 0.287 | -6.07 | 1 | 4 | 3 | -59 | -1 |
| 59 | 0.272 | 0.572 | 0.85 | 0.348 | 3 | 5 | 2 | 4 | 14 |
| 60 | 0.831 | 0.681 | 0.185 | -4.57 | 2 | 4 | 2 | -44 | -1 |
| 61 | 0.665 | 0.959 | 0.657 | 0.62 | 4 | 1 | 4 | 4 | 13 |
| 62 | 0.013 | 0.511 | 0.224 | 0.346 | 1 | 5 | 3 | 4 | 13 |
| 63 | 0.321 | 0.453 | 0.826 | 0.158 | 4 | 5 | 2 | 2 | 13 |
| 64 | 0.287 | 0.831 | 0.599 | 0.138 | 3 | 2 | 5 | 2 | 12 |
| 65 | 0.347 | 0.134 | 0.574 | 0.414 | 4 | 2 | 5 | 5 | 16 |
| 66 | 0.713 | 0.549 | 0.423 | 0.64 | 3 | 5 | 5 | 4 | 17 |
| 67 | 0.45 | 0.189 | 0.389 | -5.65 | 5 | 2 | 4 | -55 | -1 |
| 68 | 0.925 | 0.507 | 0.783 | -6.11 | 1 | 5 | 3 | -60 | -1 |
| 69 | 0.01 | 0.684 | 0.446 | 0.139 | 1 | 4 | 5 | 2 | 12 |
| 70 | 0.129 | 0.374 | 0.404 | 0.606 | 2 | 4 | 5 | 4 | 15 |
| 71 | 0.929 | 0.609 | 0.45 | 0 | 1 | 4 | 5 | 1 | -1 |
| 72 | 0.099 | 0.43 | 0.345 | 0.204 | 1 | 5 | 4 | 3 | 13 |
| 73 | 0.076 | 0.768 | 0.91 | -5.51 | 1 | 3 | 1 | -54 | -1 |
| 74 | 0.939 | 0.249 | 0.493 | 0.035 | 1 | 3 | 5 | 1 | 10 |
| 75 | 0.797 | 0.396 | 0.723 | 0.37 | 3 | 4 | 3 | 4 | 14 |
| 76 | 0.057 | 0.822 | 0.678 | -5.03 | 1 | 2 | 4 | -49 | -1 |
| 77 | 0.586 | 0.532 | 0.745 | 0.135 | 5 | 5 | 3 | 2 | 15 |
| 78 | 0.743 | 0.251 | 0.341 | -6.33 | 3 | 3 | 4 | -62 | -1 |
| 79 | 0.281 | 0.663 | 0.898 | 0.052 | 3 | 4 | 2 | 1 | 10 |
| 80 | 0.203 | 0.576 | 0.891 | -4.49 | 3 | 5 | 2 | -43 | -1 |
| | ===> final | tuftest <=== | | | | | | | |

### 3.6. reticulado_CR

### 3.6.1.- código reticulado_CR

El programa en C sigue a continuación.

```
//#include <stdio.h>
//  reticulado de [1] Gonzalez y otros GNSA33

/* Este archivo contiene un simple main() para implantar reticulado_CR
   de reticulado que usa 2 instancias de reticulao19 que es la version
   en entero sin signo de 32 bits de la ecuacion 19, en
   doble precision que implanta un reticulado acoplado..
```

```
  La ecuación 19 es:

  xn+1(i) = [(a+bxn(i-1)+cx(i+1))xn(i) + dxn(i-1) +fxn(i+1) +0.1] (mod 1)

  tomadas de [1] Gonzalez, Moreno y Guerrero, Non-invertible Transformations
  and Spaciotemporal Randomness. 2005.

  Los valores a, b, c, d, f, son constantes reales mayores que 1.
  x0(-2), x0(-1),x0(0), x0(10, x0(1) son los valores iniciales del metodo.
  izq (x0(-2)) y der (x0(2)) son 2 congruenciales lineales módulo 1, que se necesitan
  para calcular y que no estan especificados en el metodo original.

  El valor devuelto z como generado es (z =xn(-1) + x0(0) + x0(1)mod1) ya que el
  término original x0(0) no passa Tuftest.

  Se toman 2 instancias distintas reticulado1 y reticulado2, que difieren en las constantes
  y en las condiciones de borde izquierda y derecha En base a estos dos valores
   generados se construyen 2 enteros sin signo, luego la suma de estos, módulo 2**32
proporciona el valor UL generado.

   Se emplea una tabla de mezcla para almacenar los valores desde donde se los extrae al
azar.

  También tiene una rutina de reinicialización que, al azar, intercambia los valores de los
  estados de las instancias reticulado1 y reticulado2.

  programador R.A. Pastoriza       sep/2007- mayo/2008
*/

#include <stdio.h>
//////////////////////////////////////////////////////////////////////////////

        // valores iniciales de reticulado_g1
        double alfa=1.51, beta= 1.71, gama= 1.61, izq= 1.41, der= 1.91;

        // valores iniciales de reticulado_g2
        double alfa2=10.15, beta2= 11.25, gama2= 18.37, izq2= 23.54, der2= 14.78;


//////////////////////////////////////////////////////////////////////////////
   // constantes de reticulado_g1
   double aa =2.13, bb=1.45, cc=1.13 , dd= 1.25,ff= 1.15;
```

```
/////////////////////////////////////////////////////////////////////////////

   double reticulado_g1(void)
         {
          double hh;
//  eq 19

     izq = 2.364632331999018e-1 * izq + 0.287445029243826e-4; // otro GCL como borde
     izq = izq - (int)izq;      // la condicion de borde izquierda xn(-2)

     der = 1.6081379726529121e-1 *der + 0.445029243987863e-4; // otro GCL como borde
     der = der - (int)der;      // la condicion de borde derecha xn(2)

          alfa = ((aa +bb*izq + cc*beta)*alfa + dd*izq + ff*beta + 0.1);
         alfa = alfa -(int)alfa;    // xn(-1)

          beta = ((aa + bb*alfa + cc*gama)*beta + dd*alfa + ff*gama + 0.1);
     beta = beta -(int)beta;    // xn(0)

          gama = ((aa + bb*beta + cc*der)*gama + dd*beta + ff*der + 0.1);
     gama = gama - (int)gama;   // xn(1)

          hh = (alfa + beta + gama);  // modificacion del original uno solo no pasa
          hh = hh -(int)hh;

//        printf(" # en ret11 == %lf \n", hh);
          return hh;
         }
/////////////////////////////////////////////////////////////////////////////

         unsigned long reticulado1(void)
         {
                unsigned long ww;

                double reticulado_g1();

                ww = 4294967295*reticulado_g1(); // 2**32 - 1

                return ww;
         }

/////////////////////////////////////////////////////////////////////////////////

         // constantes de reticulado_g2
```

```
   double aaa =1.5, bbb=1.98, ccc=2.45 , ddd= 1.93,fff= 1.77;


/////////////////////////////////////////////////////////////////////////////

   double reticulado_g2(void)
        {// otros valores en constantes e iniciales
         double hh;
//  eq 19

      izq2 = 0.39131846278905868e-3 * izq2 + 0.287445029243826e-3; // otro GCL como
borde
     izq2 = izq2 - (int)izq2;      // la condicion de borde izquierda xn(-2)

     der2 = 1.60813797265291 21e-3 *der2 + 0.12345450292438 2686e-3; // otro GCL como
borde
     der2 = der2 - (int)der2;      // la condicion de borde derecha xn(2)

         alfa2 = ((aaa + bbb*izq2 + ccc*beta2)*alfa2 + ddd*izq2 + fff*beta2 + 0.1);
     alfa2 = alfa2 -(int)alfa2;    // xn(-1)

         beta2 = ((aaa + bbb*alfa2 + ccc*gama2)*beta2 + ddd*alfa2 + fff*gama2 + 0.1);
     beta2 = beta2 -(int)beta2;    // xn(0)

         gama2 = ((aaa + bbb*beta2 + ccc*der2)*gama2 + ddd*beta2 + fff*der2 + 0.1);
     gama2 = gama2 - (int)gama2;   // xn(1)

         hh = (alfa2 + beta2 +gama2);  // modificacion del original uno solo no pasa
         hh = hh -(int)hh;

//                printf(" # en ret12 == %lf \n\n", hh);
         return hh;
        }
/////////////////////////////////////////////////////////////////////////////

        unsigned long reticulado2(void)
        {
             unsigned long ww;

             double reticulado_g2();

             ww = 4294967295*reticulado_g2(); // 2**32 - 1

             return ww;
```

```
}

//////////////////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////////////

 unsigned long reticulado(void)
    {// componer resultado
  unsigned long w;

          unsigned long reticulado1();
  unsigned long reticulado2();

          w = (reticulado1() + reticulado2())%4294967296; // 2**32
//        printf(" w== %u \n",w);
  return w;
    }

//////////////////////////////////////////////////////////////////////////

  int minimo(int a, int b)
          {
               int m;
               m = a;
               if(b<a) m= b;
               return m;
          }
/////////////////////////////////////////////////////////////////////////

   unsigned long Tabla[128];

   void iniciar_reticulado_CR(void)

     { // iniciar reticulado_CR
          double reticulado_g1();
          double reticulado_g2() ;

          unsigned long reticulado();

  int k= 17;
          int kk1,kk2;
       int i,j;
```

```
// inicializar tabla de mezcla con 0

                    for(i=0;i<=127;i++)
                    {
                        Tabla[i] = 0;
                    }


//    calentamiento de reticulado, diferente por cada componente

//                  printf(" iniciar calentamiento \n");

                  kk1 = (int) 100*(alfa + beta + gama +izq +der); // tamanio variable del
calentamiento
                    kk2 = (int) 100*(alfa2 + beta2 + gama2 +izq2 +der2);

              kk1 = kk1&63;
                    kk1 = minimo(kk1,k);
                    if(kk1==0) kk1 = 2;

                    kk2 = kk2&63;
        kk2 = minimo(kk2,k);
                    if(kk2==0) kk2 = 2;

              for(i=1;i<=kk1;i++)
                     {// calentamiento dinámicog1
                              reticulado_g1();
                     }

                    for(j=1;j<=kk2;j++)
                     {// calentamiento dinámicog1
                              reticulado_g2();
                     }


  // llenar la tabla inicial

                    for(i=0;i<=127;i++)
                    {
         Tabla[i] = reticulado();
                    }

             }
```

```
/////////////////////////////////////////////////////////////////////////////
    int kswitch = 1;
        unsigned long generar_reticulado_CR(unsigned long z)
        {
                int i1;

                double reticulado_g1();
                double reticulado_g2();

                unsigned long reticualdo();

// generar valores
                if(kswitch ==1)
                {// alternar calculo del indice
                   i1 = (int)128*reticulado_g1();   // indice de extraccion
//                          printf(" i1== %d \n", i1);
                        kswitch = 0;
        }
                else
                {
                        i1 = (int)128*reticulado_g2();   // indice de extraccion
//                                          printf(" i1== %d \n", i1);
                   kswitch = 1;
                }

                z = Tabla[i1]; // valor extraido de la Tabla de mezcla

// reemplazar valor extraido

                Tabla[i1] = reticulado();

//              printf(" generar_reticulado_CR z== %f \n",z);

                return z;
        }

/////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
                int s=0;

            void reiniciar_reticulado_CR(unsigned long z)
                {
                 double x1_o, x2_o, x3_o, x4_o, x5_o;
```

```
                double y1_o, y2_o, y3_o, y4_o, y5_o;

//                      int limite=4;
                        int limite= 600000;

                        s = s + (z)%10;
//                          printf(" z== %u  (z)%10== %d \n", z, (z)%10);
//                          printf(" s== %d \n",s);

                        if(s>limite)
                                        {
// inicializa
//                                  printf(" *** entrando reiniciar_reticulado_CR *** \n");
//                          printf(" se paso s=  %d \n",s);
                    s= (z)%10;
//                          printf(" nuevo inicial s=  %d \n",s);

// guarda estado reticualdo1
                    x1_o = izq ;
                                        x2_o = der;
                                        x3_o = alfa;
                                        x4_o = beta;
                                        x5_o = gama;

// guarda estado reticulado2
                    y1_o = izq2 ;
                                        y2_o = der2;
                                        y3_o = alfa2;
                                        y4_o = beta2;
                                        y5_o = gama2;

// intercambia estados
// reticulado1
                                        izq =  y1_o;
                                        der =  y2_o;
                                        alfa = y3_o;
                                        beta = y4_o;
                                        gama = y5_o;
// reticulado2
                                        izq2 =  x1_o;
                                        der2 =  x2_o;
                                        alfa2 = x3_o;
                                        beta2 = x4_o;
                                        gama2 = x5_o;
```

```
//                                    printf("izq original %lf izq cambiada %lf
\n",x1_o,izq);

                              }

        }

///////////////////////////////////////////////////////////////////////////

        int vez_reticulado =1;

    unsigned long reticulado_CR()
{
    unsigned long yy;
    unsigned long z,zz;

        void iniciar_reticulado_CR();

        {
                if(vez_reticulado==1)
                {// inicializacion del generador
//                      printf(" entrando iniciar_reticulado_CR %d \n", vez_reticulado);

                        iniciar_reticulado_CR();

                        yy = generar_reticulado_CR(z);
 //             printf(" entero en generar_reticulado_CR %u \n", yy);

                        vez_reticulado =0;

                        return yy;
                }
      else
                {// generacion

                        yy = generar_reticulado_CR(z);
//            printf(" entero en generar_reticulado_CR %u \n", yy);

        reiniciar_reticulado_CR(z);

//                              printf(" estado s en reiniciar_reticualdo_CR ==>    %1u
\n",s);
```

```
                    return yy;
               }
          }
 }

//////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////

//  programa de prueba
          unsigned long main(void)
{
     int i;
     unsigned long z;

          printf("          prueba reticulado_CR \n");

          unsigned long reticulado_CR();

          for(i=1;i<=10;i++)
          {
                    z = reticulado_CR();
             printf("iter i-> %d numero en main ==> %u \n\n",i, z);

          }

          printf("\n          en 10 iteraciones  x10 es 3112424930 \n");
          printf("iter =  %d el numero en main es ==>   %u \n",i-1,z);

     return (z);
}

//////////////////////////////////////////////////////////////////////////////
```

Resultados Tuftest

```
++++++++++++++++++++++++++++++++++++++++++
Birthday spacings test: 4096 birthdays, 2^32 days in year
        Table of Expected vs. Observed counts:
Duplicates   0     1     2     3     4     5     6     7     8     9   >=10
Expected   91.6 366.3 732.6 976.8 976.8 781.5 521.0 297.7 148.9  66.2  40.7
Observed    76   381   770   990   984   738   504   305   149    68    35
(O-E)^2/E   2.6   0.6   1.9   0.2   0.1   2.4   0.6   0.2   0.0   0.1   0.8
```

Birthday Spacings: Sum(O-E)^2/E= 9.365, p= 0.502

Euclid's algorithm:
p-value, steps to gcd: 0.506762
p-value, dist. of gcd's: 0.178193

Gorilla test for 2^26 bits, positions 0 to 31:
Note: lengthy test---for example, ~20 minutes for 850MHz PC
Bits 0 to 7---> 0.292 0.389 0.621 0.785 0.013 0.736 0.876 0.373
Bits 8 to 15---> 0.949 0.034 0.933 0.794 0.378 0.684 0.144 0.331
Bits 16 to 23---> 0.155 0.277 0.488 0.639 0.706 0.711 0.688 0.478
Bits 24 to 31---> 0.743 0.180 0.510 0.796 0.045 0.903 0.162 0.400
KS test for the above 32 p values: 0.043
Press any key to continue
++++++++++++++++++++++++++++++++++++++++++++++++

### 3.6.2.- resultados de pruebas Tuftest a reticulado_CR

| | | tuftest | _reticulado_CR | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **cumpleaños** | **gcd Euc** | **lides** | **gorila** | | | | | |
| 1 | **0.502** | **0.506762** | **0.178193** | **0.043** | **5** | **5** | **2** | **1** | **13** |
| 2 | **0.965** | **0.637602** | **0.737995** | **0.185** | **1** | **4** | **3** | **2** | **10** |
| 3 | **0.330** | **0.844408** | **0.504934** | **0.051** | **4** | **2** | **5** | **1** | **12** |
| 4 | **0.246** | **0.290433** | **0.360392** | **0.507** | **3** | **3** | **4** | **5** | **15** |
| 5 | **0.551** | **0.886353** | **0.212708** | **0.551** | **5** | **2** | **3** | **5** | **15** |
| 6 | **0.900** | **0.609765** | **0.701701** | **0.390** | **1** | **4** | **3** | **4** | **12** |
| 7 | **0.834** | **0.335296** | **0.604291** | **-4.703** | **2** | **4** | **4** | **-46** | **-1** |
| 8 | **0.051** | **0.461799** | **0.148487** | **-4.555** | **1** | **5** | **2** | **-44** | **-1** |
| 9 | **0.112** | **0.307397** | **0.582312** | **0.455** | **2** | **4** | **5** | **5** | **16** |
| 10 | **0.178** | **0.904735** | **0.376559** | **0.019** | **2** | **1** | **4** | **1** | **8** |
| 11 | **0.129** | **0.562240** | **0.099791** | **0.598** | **2** | **5** | **1** | **5** | **13** |
| 12 | **0.571** | **0.715505** | **0.601316** | **0.112** | **5** | **3** | **4** | **2** | **14** |
| 13 | **0.491** | **0.215582** | **0.493971** | **0.092** | **5** | **3** | **5** | **1** | **14** |
| 14 | **0.835** | **0.421315** | **0.382886** | **0.484** | **2** | **5** | **4** | **5** | **16** |
| 15 | **0.536** | **0.150093** | **0.291613** | **-5.107** | **5** | **2** | **3** | **-50** | **-1** |
| 16 | **0.770** | **0.224051** | **0.668800** | **0.021** | **3** | **3** | **4** | **1** | **11** |
| 17 | **0.582** | **0.737053** | **0.675033** | **0.206** | **5** | **3** | **4** | **3** | **15** |
| 18 | **0.150** | **0.546450** | **0.280684** | **0.491** | **2** | **5** | **3** | **5** | **15** |
| 19 | **0.971** | **0.206513** | **0.181722** | **-4.239** | **1** | **3** | **2** | **-41** | **-1** |
| 20 | **0.836** | **0.257180** | **0.572423** | **0.085** | **2** | **3** | **5** | **1** | **11** |
| 21 | **0.963** | **0.610048** | **0.497829** | **0.365** | **1** | **4** | **5** | **4** | **14** |
| 22 | **0.726** | **0.467849** | **0.419868** | **-4.644** | **3** | **5** | **5** | **-45** | **-1** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 23 | **0.795** | **0.449416** | **0.542561** | **0.526** | **3** | **5** | **5** | **5** | **18** |
| 24 | **0.656** | **0.491648** | **0.291977** | **0.260** | **4** | **5** | **3** | **3** | **15** |
| 25 | **0.615** | **0.919753** | **0.892253** | **0.255** | **4** | **1** | **2** | **3** | **10** |
| 26 | **0.421** | **0.399013** | **0.249738** | **0.003** | **5** | **4** | **3** | **1** | **-1** |
| 27 | **0.765** | **0.464172** | **0.653152** | **-5.276** | **3** | **5** | **4** | **-51** | **-1** |
| 28 | **0.709** | **0.246637** | **0.432666** | **0.179** | **3** | **3** | **5** | **2** | **13** |
| 29 | **0.674** | **0.776317** | **0.166483** | **0.623** | **4** | **3** | **2** | **4** | **13** |
| 30 | **0.420** | **0.508698** | **0.228472** | **0.102** | **5** | **5** | **3** | **2** | **15** |
| 31 | **0.860** | **0.794533** | **0.305448** | **-5.858** | **2** | **3** | **4** | **-57** | **-1** |
| 32 | **0.833** | **0.665493** | **0.865791** | **0.088** | **2** | **4** | **2** | **1** | **9** |
| 33 | **0.985** | **0.276867** | **0.190117** | **0.052** | **1** | **3** | **2** | **1** | **7** |
| 34 | **0.072** | **0.182321** | **0.761818** | **0.209** | **1** | **2** | **3** | **3** | **9** |
| 35 | **0.104** | **0.363783** | **0.440229** | **0.273** | **2** | **4** | **5** | **3** | **14** |
| 36 | **0.527** | **0.930760** | **0.703350** | **0.482** | **5** | **1** | **3** | **5** | **14** |
| 37 | **0.162** | **0.711995** | **0.038687** | **0.494** | **2** | **3** | **1** | **5** | **11** |
| 38 | **0.646** | **0.228396** | **0.632678** | **-4.534** | **4** | **3** | **4** | **-44** | **-1** |
| 39 | **0.855** | **0.738210** | **0.665838** | **-4.874** | **2** | **3** | **4** | **-47** | **-1** |
| 40 | **0.124** | **0.341587** | **0.041837** | **0.392** | **2** | **4** | **1** | **4** | **11** |
| 41 | **0.739** | **0.011061** | **0.197897** | **0.354** | **3** | **1** | **2** | **4** | **10** |
| 42 | **0.191** | **0.262672** | **0.501199** | **0.185** | **2** | **3** | **5** | **2** | **12** |
| 43 | **0.397** | **0.517884** | **0.362466** | **-4.441** | **4** | **5** | **4** | **-43** | **-1** |
| 44 | **0.221** | **0.597653** | **0.841461** | **-4.391** | **3** | **5** | **2** | **-42** | **-1** |
| 45 | **0.294** | **0.738265** | **0.782117** | **-4.386** | **3** | **3** | **3** | **-42** | **-1** |
| 46 | **0.357** | **0.610404** | **0.754589** | **-6.067** | **4** | **4** | **3** | **-59** | **-1** |
| 47 | **0.756** | **0.272076** | **0.638558** | **0.074** | **3** | **3** | **4** | **1** | **11** |
| 48 | **0.576** | **0.516949** | **0.476100** | **0.197** | **5** | **5** | **5** | **2** | **17** |
| 49 | **0.959** | **0.573646** | **0.445041** | **-4.940** | **1** | **5** | **5** | **-48** | **-1** |
| 50 | **0.958** | **0.562756** | **0.051128** | **-5.417** | **1** | **5** | **1** | **-53** | **-1** |
| 51 | **0.330** | **0.580679** | **0.179133** | **0.059** | **4** | **5** | **2** | **1** | **12** |
| 52 | **0.474** | **0.571147** | **0.449453** | **0.163** | **5** | **5** | **5** | **2** | **17** |
| 53 | **0.124** | **0.274569** | **0.664787** | **0.218** | **2** | **3** | **4** | **3** | **12** |
| 54 | **0.899** | **0.739590** | **0.293696** | **0.345** | **2** | **3** | **3** | **4** | **12** |
| 55 | **0.168** | **0.155782** | **0.440205** | **0.126** | **2** | **2** | **5** | **2** | **11** |
| 56 | **0.710** | **0.212754** | **0.661818** | **0.333** | **3** | **3** | **4** | **4** | **14** |
| 57 | **0.447** | **0.635695** | **0.648560** | **-5.556** | **5** | **4** | **4** | **-54** | **-1** |
| 58 | **0.931** | **0.195307** | **0.127948** | **0.274** | **1** | **2** | **2** | **3** | **8** |
| 59 | **0.993** | **0.129946** | **0.315641** | **0.481** | **1** | **2** | **4** | **5** | **-1** |
| 60 | **0.153** | **0.717253** | **0.855731** | **-4.832** | **2** | **3** | **2** | **-47** | **-1** |
| 61 | **0.384** | **0.729929** | **0.806580** | **0.408** | **4** | **3** | **2** | **5** | **14** |
| 62 | **0.143** | **0.314556** | **0.277416** | **-7.189** | **2** | **4** | **3** | **-70** | **-1** |
| 63 | **0.146** | **0.945521** | **0.596268** | **-5.259** | **2** | **1** | **5** | **-51** | **-1** |
| 64 | **0.731** | **0.676885** | **0.128729** | **0.222** | **3** | **4** | **2** | **3** | **12** |
| 65 | **0.447** | **0.801305** | **0.463203** | **0.255** | **5** | **2** | **5** | **3** | **15** |
| 66 | **0.673** | **0.625608** | **0.666855** | **0.425** | **4** | **4** | **4** | **5** | **17** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 67 | 0.479 | 0.321088 | 0.458193 | 0.528 | 5 | 4 | 5 | 5 | 19 |
| 68 | 0.138 | 0.360479 | 0.303445 | -5.471 | 2 | 4 | 4 | -53 | -1 |
| 69 | 0.503 | 0.789525 | 0.104134 | -5.273 | 5 | 3 | 2 | -51 | -1 |
| 70 | 0.492 | 0.800149 | 0.202413 | 0.088 | 5 | 2 | 3 | 1 | 11 |
| 71 | 0.191 | 0.375677 | 0.637372 | -5.468 | 2 | 4 | 4 | -53 | -1 |
| 72 | 0.485 | 0.178604 | 0.036278 | -4.169 | 5 | 2 | 1 | -40 | -1 |
| 73 | 0.500 | 0.493571 | 0.695408 | -5.216 | 6 | 5 | 4 | -51 | -1 |
| 74 | 0.113 | 0.652604 | 0.465430 | -4.924 | 2 | 4 | 5 | -48 | -1 |
| 75 | 0.912 | 0.332758 | 0.715346 | -5.154 | 1 | 4 | 3 | -50 | -1 |
| 76 | 0.874 | 0.281718 | 0.647423 | 0.468 | 2 | 3 | 4 | 5 | 14 |
| 77 | 0.871 | 0.960193 | 0.694481 | -4.233 | 2 | 1 | 4 | -41 | -1 |
| 78 | 0.555 | 0.366577 | 0.417708 | -5.435 | 5 | 4 | 5 | -53 | -1 |
| 79 | 0.001 | 0.269206 | 0.706010 | -4.727 | 1 | 3 | 3 | -46 | -1 |
| 80 | 0.845 | 0.581896 | 0.558534 | 0.008 | 2 | 5 | 5 | 1 | -1 |
| 81 | 0.034 | 0.252129 | 0.659104 | 0.158 | 1 | 3 | 4 | 2 | 10 |
| 82 | 0.107 | 0.628044 | 0.776349 | -5.887 | 2 | 4 | 3 | -57 | -1 |
| 83 | 0.045 | 0.252640 | 0.614784 | -5.626 | 1 | 3 | 4 | -55 | -1 |
| 84 | 0.542 | 0.531617 | 0.337102 | 0.238 | 5 | 5 | 4 | 3 | 17 |
| 85 | 0.680 | 0.367271 | 0.622705 | 0.091 | 4 | 4 | 4 | 1 | 13 |
| 86 | 0.094 | 0.323543 | 0.484944 | 0.023 | 1 | 4 | 5 | 1 | 11 |
| 87 | 0.684 | 0.508838 | 0.483164 | 0.439 | 4 | 5 | 5 | 5 | 19 |
| 88 | 0.670 | 0.301338 | 0.310121 | -5.583 | 4 | 4 | 4 | -54 | -1 |
| 89 | 0.332 | 0.803350 | 0.887097 | 0.555 | 4 | 2 | 2 | 5 | 13 |
| 90 | 0.265 | 0.982595 | 0.120390 | 0.394 | 3 | 1 | 2 | 4 | 10 |
| 91 | 0.631 | 0.299572 | 0.632335 | -5.350 | 4 | 3 | 4 | -52 | -1 |
| 92 | 0.460 | 0.840485 | 0.772800 | 0.060 | 5 | 2 | 3 | 1 | 11 |
| 93 | 0.152 | 0.151566 | 0.772586 | -6.068 | 2 | 2 | 3 | -59 | -1 |
| 94 | 0.763 | 0.426006 | 0.587148 | 0.570 | 3 | 5 | 5 | 5 | 18 |
| 95 | 0.046 | 0.380050 | 0.377991 | -5.286 | 1 | 4 | 4 | -51 | -1 |
| 96 | 0.379 | 0.249204 | 0.647471 | 0.447 | 4 | 3 | 4 | 5 | 16 |
| 97 | 0.579 | 0.432490 | 0.252126 | 0.161 | 5 | 5 | 3 | 2 | 15 |
| 98 | 0.058 | 0.580795 | 0.822639 | 0.015 | 1 | 5 | 2 | 1 | 9 |
| 99 | 0.896 | 0.631004 | 0.693676 | 0.066 | 2 | 4 | 4 | 1 | 11 |
| 100 | 0.209 | 0.175083 | 0.650003 | -5.088 | 3 | 2 | 4 | -49 | -1 |
| 101 | 0.396 | 0.548237 | 0.224342 | 0.253 | 4 | 5 | 3 | 3 | 15 |
| 102 | 0.076 | 0.186866 | 0.205684 | 0.581 | 1 | 2 | 3 | 5 | 11 |
| 103 | 0.821 | 0.802744 | 0.798918 | -4.253 | 2 | 2 | 3 | -41 | -1 |
| 104 | 0.696 | 0.403112 | 0.563739 | 0.077 | 4 | 5 | 5 | 1 | 15 |
| 105 | 0.876 | 0.674163 | 0.493347 | 0.282 | 2 | 4 | 5 | 3 | 14 |
| 106 | 0.994 | 0.983756 | 0.450043 | -6.080 | 1 | 1 | 5 | -59 | -1 |
| 107 | 0.553 | 0.898790 | 0.293420 | 0.045 | 5 | 2 | 3 | 1 | 11 |
| 108 | 0.221 | 0.508515 | 0.830541 | 0.584 | 3 | 5 | 2 | 5 | 15 |
| 109 | 0.788 | 0.216960 | 0.757768 | 0.437 | 3 | 3 | 3 | 5 | 14 |
| 110 | 0.068 | 0.147000 | 0.439547 | 0.594 | 1 | 2 | 5 | 5 | 13 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 111 | 0.348 | 0.733565 | 0.320469 | -5.226 | 4 | 3 | 4 | -51 | -1 |
| 112 | 0.430 | 0.392482 | 0.375445 | 0.431 | 5 | 4 | 4 | 5 | 18 |
| 113 | 0.586 | 0.220225 | 0.459334 | -4 /* double AA=151.4751; mejores double BB=173.4789; .250 | 5 | 3 | 5 | -41 | -1 |
| 114 | 0.907 | 0.062068 | 0.615625 | 0.145 | 1 | 1 | 4 | 2 | 8 |
| 115 | 0.739 | 0.344505 | 0.220214 | -6.275 | 3 | 4 | 3 | -61 | -1 |
| 116 | 0.480 | 0.747763 | 0.760605 | 0.189 | 5 | 3 | 3 | 2 | 13 |
| 117 | 0.999 | 0.752863 | 0.171930 | 0.011 | 1 | 3 | 2 | 1 | -1 |
| 118 | 0.180 | 0.582261 | 0.458385 | -4.567 | 2 | 5 | 5 | -44 | -1 |
| 119 | 0.687 | 0.094583 | 0.151284 | -7.031 | 4 | 1 | 2 | -69 | -1 |
| 120 | 0.963 | 0.174200 | 0.830382 | 0.581 | 1 | 2 | 2 | 5 | 10 |
| 121 | 0.987 | 0.555093 | 0.209387 | 0.417 | 1 | 5 | 3 | 5 | 14 |
| 122 | 0.938 | 0.858060 | 0.419764 | 0.420 | 1 | 2 | 5 | 5 | 13 |
| 123 | 0.172 | 0.332842 | 0.508988 | 0.255 | 2 | 4 | 5 | 3 | 14 |
| 124 | 0.204 | 0.498420 | 0.362734 | -5.643 | 3 | 5 | 4 | -55 | -1 |
| 125 | 0.881 | 0.864496 | 0.323948 | -4.841 | 2 | 2 | 4 | -47 | -1 |
| 126 | 0.351 | 0.472284 | 0.470426 | 0.360 | 4 | 5 | 5 | 4 | 18 |
| 127 | 0.970 | 0.346304 | 0.760300 | -4.660 | 1 | 4 | 3 | -45 | -1 |
| | ===> final | tuftest < | === | | | | | | |

## 3.7. FF_CR

### 3.6.1.- código FF_CR

El programa en C sigue a continuación.

```
#include <stdio.h>
#include <stdlib.h>
/*
FF_CR es la version en enteros grandes criptograficamente segura de FF.
FF en doble precision esta inspirado en porfiado4.
FF_CR_66 tiene una tabla de mezcla incluida

El esquema de calculo de FF_CR es;

  FF_CR
     |  inicializar
     |---init_FF_CR  inicializa FF_CR
     |        |
     |        |--- FF     calcula nuevo valor en doble precision
     |
     |---FF_UL_CR          calcula valor en entero grande
     |        |
     |        |--- FF     calcula nuevo valor en doble precision
     |
     |  generar
     |---FF_UL_CR          calcula valor en entero grande
     |        |
     |        |--- FF     calcula nuevo valor en doble precision
     |
     |--- reiniciar_FF_CR   reinicializa estado del generador

 programador: R.A. Pastoriza   junio 2008
             UCV. Ciencias. Computacion. CIOMMA.
 */
/*//////////////////////////////////////////////////////////////////////////*/

/*   valores iniciales FF  */
          static double xx=1.54321, yy=2.56789;

    static double FFx=5.0, FFy=5.5;

/*//////////////////////////////////////////////////////////////////////////*/
```

```
/*     unsigned long calls = 0;   */ /*  cuenta llamadas a FF()   */

/*
FF es un generador derivado de porfiado4. Se realizan las operaciones en doble precision.

La transformacion  no lineal base es;

      x= AA/(x + (1.0/(Fy+1.0) + 1.75432198))
      y= AA/(y + (1.0/(Fx+1.0) + 3.14159265))

donde AA y BB son constantes reales.

Su esquema de calculo es el siguiente:
   1.- Tomar las mantisas de x y y en doble precision
   2.- Llevar esas mantisas a enteros grandes n1 y n2 (32bits)
   3.- Hacer una rotación circular de los enteros n1 y n2
   4.- Llevar esos enteros rotados a doble  precision
   5.- Calcular x y y con la transformación no lineal
   6.- Calcular Fx y Fy descartando las 3 primeras posiciones decimales de x y y
   7.- Devolver Fx

*/

       long unsigned int M = 4294967295UL;  /* 2**32 -1   */

           double FF(void) /* generacion de valores aleatorios doble precision   */
           {
                      /* AA y BB constantes metodo */
           /*         double AA=  151.4751;  mejores
                      double BB=  173.4789;        */

                      double AA= 113.897531;
                      double BB= 145.347987;

                      double  w;

                      unsigned long int n1,n2;

                      int kh;

/*                     printf(" entrando en FF xx== %lf yy== %lf \n",xx,yy);  */
/*               printf(" FFx== %20.16e FFy== %20.16e \n", FFx,FFy);   */

/*                    calls = calls +1;   */
```

```
        xx = xx -(int)xx;  /* mantisa de xx   */
        yy = yy -(int)yy;
/*        printf(" entrando FF() xx== %20.16e FFx== %20.16e \n",xx,FFx);  */
        n1 = (unsigned long int)((unsigned long int)M * (double)xx);  /* xx --> entero  */
        n2 = (unsigned long int)((unsigned long int)M * (double) yy);
        kh = 11;    /*  constante rotacion circular n1   */

/*              printf(" antes girar  n1 == %u kh == %i\n",n1,kh);   */
             n1 = ((n1>>kh) ^ ((n1<<(32-kh))));  /* rotacion circular de n1 */
/*                  printf(" despues girar n1 == %u \n",n1);  */

                 kh = kh+ 4;  /*  constante rotacion circular n2   */
/*              printf(" antes girar  n2 == %u kh == %i\n",n2,kh);   */
             n2 = ((n2>>kh) ^ ((n2<<(32-kh)))); /* rotacion circular de n2 */
/*                  printf(" despues girar n2 == %u \n",n2);  */

                 xx = n1*2.3283064365386962e-10; /* conversion a double (1/2**32) */
                   /*   2.3283064365386962e-10 con 16 digitos  */
                 yy = n2*2.3283064365386962e-10;
/*        printf(" volviendo a enteros xx== %20.18e  yy== %20.18e \n",xx,yy);  */

                 xx = AA/((double)xx + 1.0/((double)FFy + 1.0) + 1.75432198);   /* nueva
transformacion no lineal */
        yy = BB/((double)yy + 1.0/((double)FFx + 1.0) + 3.14159265);
/*                   printf("  en  FF_CR_55  luego  cuenta    xx==  %20.16e  yy==  %20.16e
\n",xx,yy);   */

        FFx = (1000*xx) - (int)(1000*xx); /* FFx doble descartando 3 posiciones de xx */
                 FFy = (1000*yy) - (int)(1000*yy);
/*                  printf(" FFx== %20.16e FFy== %20.16e \n", FFx,FFy);  */

        w = FFx;

/*                  printf(" saliendo en FF w== %20.16e \n\n\n",w);  */
/*        printf(" llamadas== %u \n", calls);  */
                 return w;
          }
/* ///////////////////////////////////////////////////////////////////////////////// */

/* ////////////////////////////////////////////////////////////////////////////////// */
        unsigned long int Tabla[128];

  void init_FF_CR(double yy3,double yy4) /* inicializacion del generador entero   */
```

```
{
  double w;

  int i, k;
/*                  printf(" en init_FF_CR entrando yy3==  %lf yy4== %lf \n",yy3,yy4);  */
  (xx=yy3,yy=yy4);
/*                  printf(" en init_FF_CR reasignando  xx==  %lf yy== %lf \n",xx,yy);  */

  k = (int)(10*(xx+yy)); /* cuantas veces calcular FF() en calentamiento   */

/*   printf(" en init_FF_CR veces k==>> %d \n",k);   */
  if(k >= 6) k =6;
  for(i=0; i<= k; i++) /* inicializacion variable   */
  {
        w = FF();
/*    printf("i== %d xx== %lf yy== %lf w==>> %lf\n\n",i,xx,yy,w);   */
  }
/*                    printf("  en  init_FF_CR  saliendo  xx==  %lf  yy==  %lf    w==    %lf
\n",xx,yy,w);  */

  for(i=0; i<=127;i++)
   {   /* inicializar tabla de mezcla   */
        w = FF();
        Tabla[i] = (unsigned long int)((unsigned long int)M * (double)w);;
   }
        return;
}

/* //////////////////////////////////////////////////////////////////////////// */

                 unsigned long int s=0;

            void reiniciar_FF_CR(unsigned long z) /* reinicializacion del generador entero
*/
                 {
                  double x1_o, x2_o;

/*                unsigned long int limite=4;    */  /* limite para reinicializar solo pruebas   */
                  unsigned long int limite= 600000;

                  s = s + (z)%10;
/*                    printf(" z== %u  (z)%10== %d \n", z, (z)%10); */
/*                    printf(" s== %d \n",s);    */
```

```
                    if(s>limite)
                                                 {
/* inicializa  */
/*                              printf(" *** entrando reiniciar_rotar_bits *** \n");  */
/*                    printf(" se paso s=  %d \n",s);   */
                  s= (z)%10;

/*                                               printf(" nuevo inicial s=  %d \n",s);  */

/*                                                      printf("x1==    %lf,    x4==    %lf,x==%lf,y==
%lf\n",x1,x4,x,y);  */
/* guarda estado  */
                  x1_o = xx;
/*                                               printf(" x1_o== %lf \n", x1_o);   */
                                                 x2_o = yy;
/*                                               printf(" x2_o== %lf \n", x2_o);   */

/*                                                   printf("xx   original==   %lf   yy   original==   %lf
\n",x1_o,x2_o);  */

/* cambia estado   */


                                                 xx  = x2_o;
                  yy  = x1_o;
/*                                                 printf("xx cambiado==> %lf yy cambiado==> %lf
\n",xx,yy);   */

                                                 }
          }

/* ////////////////////////////////////////////////////////////////////////////////// */

   unsigned long int FF_UL_CR(void) /* generacion de enteros aleatorios   */
   {
          unsigned long int entra,sale;
     int indice;
     double uu;
/*             printf("   en   FF_UL_CR   entrando   xx==   %lf   FFx==   %lf   FFy==   %lf
\n\n",xx,FFx,FFy);  */

      uu = FF();
      entra =(unsigned long int)((unsigned long int)M * (double)uu);
      indice = (int)(128*(double)FFy);
```

```
        sale = Tabla[indice];
        Tabla[indice] = entra;

/*          printf(" en FF_UL_CR saliendo sale_UL== %u entra== %u indice== %d \n\n",
sale,entra,indice);   */
            return sale;
    }

/* //////////////////////////////////////////////////////////////////////// */

/* ///////////////////////////////////////////////////////////////////////// */

        int vez = 1;
    unsigned long int FF_CR(void) /* generador aleatorio criptograficamente resistente   */
          {
                unsigned long int u;

                if(vez==1)  /* inicializar FF_CR   */
                {
          init_FF_CR(xx,yy);
                  u = FF_UL_CR();
/*       printf(" vez==  %d en FF_CR u==  %u  \n",vez, u);  */
              vez = 2;
                }
                else
                {          /* generar FF_CR  */
          u = FF_UL_CR();
/*        printf(" vez == %d en FF_CR     u== %u   \n",vez, u);  */
                  reiniciar_FF_CR(u);
                }
                return u;
          }

/* /////////////////////////////////////////////////////////////////////// */


/*  programa de prueba    */

        int main()
{
      int i;
      int nn;

            unsigned long n;
```

```
            unsigned long FF_CR();

            printf("         prueba FF_CR_55 Dev-C++4.9.9.2 \n\n");
            for(i=1;i<=10;i++)
            {
         n = FF_CR();
         nn = n;
               printf("iter i-> %d numero entero grande en main ==> %u \n\n",i, n);
/*      printf("iter i-> %d numero entero en main ==> %d \n\n",i, nn);   */

            }

            printf("           en 10 iteraciones  x10 es       3238666448         \n");
            printf("iter =  %d el numero entero en main es ==>   %u \n",i-1,n);
      system("PAUSE");

        return 0;
}

/* //////////////////////////////////////////////////////////////////////// */
```

### 3.7. 2.- resultados de pruebas Tuftest a FF_CR

| | | tuftest | _FF_CR_66 | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **cumpleaños** | **gcd Euc** | **lides** | **gorila** | | | | | |
| 1 | **0.608** | **0.319527** | **0.740167** | **0.449** | **4** | **4** | **3** | **5** | **16** |
| 2 | **0.372** | **0.300275** | **0.836640** | **0.092** | **4** | **4** | **2** | **1** | **11** |
| 3 | **0.947** | **0.536465** | **0.338817** | **0.366** | **1** | **5** | **4** | **4** | **14** |
| 4 | **0.207** | **0.862012** | **0.547368** | **0.594** | **3** | **2** | **5** | **5** | **15** |
| 5 | **0.865** | **0.110987** | **0.078871** | **0.742** | **2** | **2** | **1** | **3** | **8** |
| 6 | **0.029** | **0.637986** | **0.846898** | **0.289** | **1** | **4** | **2** | **3** | **10** |
| 7 | **0.785** | **0.530051** | **0.583231** | **0.982** | **3** | **5** | **5** | **1** | **14** |
| 8 | **0.546** | **0.748641** | **0.436818** | **0.516** | **5** | **3** | **5** | **5** | **18** |
| 9 | **0.123** | **0.830484** | **0.603292** | **0.356** | **2** | **2** | **4** | **4** | **12** |
| 10 | **0.089** | **0.369210** | **0.483824** | **0.275** | **1** | **4** | **5** | **3** | **13** |
| 11 | **0.340** | **0.420819** | **0.408261** | **0.200** | **4** | **5** | **5** | **3** | **17** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 12 | **0.308** | **0.521480** | **0.732125** | **0.217** | **4** | **5** | **3** | **3** | **15** |
| 13 | **0.686** | **0.782790** | **0.445981** | **0.404** | **4** | **3** | **5** | **5** | **17** |
| 14 | **0.203** | **0.158574** | **0.058105** | **0.927** | **3** | **2** | **1** | **1** | **7** |
| 15 | **0.925** | **0.736649** | **0.427038** | **0.264** | **1** | **3** | **5** | **3** | **12** |
| 16 | **0.069** | **0.518101** | **0.500516** | **0.091** | **1** | **5** | **5** | **1** | **12** |
| 17 | **0.819** | **0.702200** | **0.574201** | **0.105** | **2** | **3** | **5** | **2** | **12** |
| 18 | **0.114** | **0.449768** | **0.555443** | **0.745** | **2** | **5** | **5** | **3** | **15** |
| 19 | **0.942** | **0.284154** | **0.726406** | **0.251** | **1** | **3** | **3** | **3** | **10** |
| 20 | **0.809** | **0.578472** | **0.274582** | **0.289** | **2** | **5** | **3** | **3** | **13** |
| 21 | **0.355** | **0.924204** | **0.387329** | **0.741** | **4** | **1** | **4** | **3** | **12** |
| 22 | **0.587** | **0.261137** | **0.353396** | **0.260** | **5** | **3** | **4** | **3** | **15** |
| 23 | **0.926** | **0.198780** | **0.742681** | **0.895** | **1** | **2** | **3** | **2** | **8** |
| 24 | **0.007** | **0.201984** | **0.015067** | **0.078** | **1** | **3** | **1** | **1** | **-1** |
| 25 | **0.531** | **0.195788** | **0.753777** | **0.936** | **5** | **2** | **3** | **1** | **11** |
| 26 | **0.049** | **0.124459** | **0.481795** | **0.860** | **1** | **2** | **5** | **2** | **10** |
| 27 | **0.942** | **0.301183** | **0.983958** | **0.033** | **1** | **4** | **1** | **1** | **7** |
| 28 | **0.551** | **0.152684** | **0.601989** | **0.002** | **5** | **2** | **4** | **1** | **-1** |
| 29 | **0.370** | **0.757097** | **0.480193** | **0.496** | **4** | **3** | **5** | **5** | **17** |
| 30 | **0.445** | **0.873145** | **0.378055** | **0.982** | **5** | **2** | **4** | **1** | **12** |
| 31 | **0.892** | **0.824179** | **0.818240** | **0.417** | **2** | **2** | **2** | **5** | **11** |
| 32 | **0.292** | **0.218535** | **0.204879** | **0.221** | **3** | **3** | **3** | **3** | **12** |
| 33 | **0.109** | **0.333303** | **0.627513** | **0.403** | **2** | **4** | **4** | **5** | **15** |
| 34 | **0.802** | **0.456294** | **0.430214** | **0.547** | **2** | **5** | **5** | **5** | **17** |
| 35 | **0.854** | **0.355986** | **0.320798** | **0.496** | **2** | **4** | **4** | **5** | **15** |
| 36 | **0.853** | **0.785357** | **0.426520** | **0.396** | **2** | **3** | **5** | **4** | **14** |
| 37 | **0.991** | **0.470645** | **0.305529** | **0.864** | **1** | **5** | **4** | **2** | **-1** |
| 38 | **0.534** | **0.915497** | **0.637287** | **0.449** | **5** | **1** | **4** | **5** | **15** |
| 39 | **0.848** | **0.225158** | **0.770154** | **0.367** | **2** | **3** | **3** | **4** | **12** |
| 40 | **0.351** | **0.393533** | **0.976304** | **0.942** | **4** | **4** | **1** | **1** | **10** |
| 41 | **0.919** | **0.685630** | **0.956991** | **0.378** | **1** | **4** | **1** | **4** | **10** |
| 42 | **0.111** | **0.557971** | **0.356605** | **0.382** | **2** | **5** | **4** | **4** | **15** |
| 43 | **0.896** | **0.569961** | **0.323722** | **0.255** | **2** | **5** | **4** | **3** | **14** |
| 44 | **0.370** | **0.273663** | **0.439718** | **0.691** | **4** | **3** | **5** | **4** | **16** |
| 45 | **0.401** | **0.687733** | **0.749935** | **0.706** | **5** | **4** | **3** | **3** | **15** |
| 46 | **0.161** | **0.590349** | **0.367627** | **0.790** | **2** | **5** | **4** | **3** | **14** |
| 47 | **0.969** | **0.271292** | **0.545794** | **0.018** | **1** | **3** | **5** | **1** | **10** |
| 48 | **0.363** | **0.212379** | **0.548198** | **0.482** | **4** | **3** | **5** | **5** | **17** |
| 49 | **0.412** | **0.304934** | **0.541652** | **0.555** | **5** | **4** | **5** | **5** | **19** |
| 50 | **0.548** | **0.176996** | **0.443538** | **0.442** | **5** | **2** | **5** | **5** | **17** |
| 51 | **0.630** | **0.272158** | **0.827429** | **0.087** | **4** | **3** | **2** | **1** | **10** |
| 52 | **0.148** | **0.657999** | **0.595900** | **0.867** | **2** | **4** | **5** | **2** | **13** |
| 53 | **0.363** | **0.807162** | **0.526719** | **0.303** | **4** | **2** | **5** | **4** | **15** |
| 54 | **0.328** | **0.524701** | **0.434369** | **0.651** | **4** | **5** | **5** | **4** | **18** |
| 55 | **0.623** | **0.524798** | **0.759011** | **0.763** | **4** | **5** | **3** | **3** | **15** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 56 | **1.000** | **0.532692** | **0.677239** | **0.511** | **0** | **5** | **4** | **5** | **-1** |
| 57 | **0.515** | **0.464522** | **0.404975** | **0.597** | **5** | **5** | **5** | **5** | **20** |
| 58 | **0.840** | **0.566370** | **0.547063** | **0.410** | **2** | **5** | **5** | **5** | **17** |
| 59 | **0.391** | **0.223158** | **0.478113** | **0.840** | **4** | **3** | **5** | **2** | **14** |
| 60 | **0.560** | **0.419875** | **0.394228** | **0.160** | **5** | **5** | **4** | **2** | **16** |
| 61 | **0.952** | **0.698935** | **0.362369** | **0.666** | **1** | **4** | **4** | **4** | **13** |
| 62 | **0.024** | **0.421890** | **0.523256** | **0.827** | **1** | **5** | **5** | **2** | **13** |
| 63 | **0.822** | **0.875467** | **0.450077** | **0.330** | **2** | **2** | **5** | **4** | **13** |
| 64 | **0.130** | **0.333340** | **0.523270** | **0.044** | **2** | **4** | **5** | **1** | **12** |
| 65 | **0.545** | **0.386641** | **0.149727** | **0.771** | **5** | **4** | **2** | **3** | **14** |
| 66 | **0.795** | **0.378154** | **0.339409** | **0.931** | **3** | **4** | **4** | **1** | **12** |
| 67 | **0.306** | **0.309021** | **0.327104** | **0.049** | **4** | **4** | **4** | **1** | **13** |
| 68 | **0.432** | **0.796567** | **0.923622** | **0.958** | **5** | **3** | **1** | **1** | **10** |
| 69 | **0.548** | **0.729160** | **0.326323** | **0.730** | **5** | **3** | **4** | **3** | **15** |
| 70 | **0.919** | **0.676702** | **0.845733** | **0.799** | **1** | **4** | **2** | **3** | **10** |
| 71 | **0.731** | **0.446151** | **0.565192** | **0.469** | **3** | **5** | **5** | **5** | **18** |
| 72 | **0.600** | **0.857456** | **0.536708** | **0.279** | **4** | **2** | **5** | **3** | **14** |
| 73 | **0.402** | **0.404092** | **0.164264** | **0.734** | **5** | **5** | **2** | **3** | **15** |
| 74 | **0.644** | **0.534814** | **0.771979** | **0.033** | **4** | **5** | **3** | **1** | **13** |
| 75 | **0.624** | **0.189889** | **0.336870** | **0.851** | **4** | **2** | **4** | **2** | **12** |
| 76 | **0.245** | **0.820563** | **0.378314** | **0.969** | **3** | **2** | **4** | **1** | **10** |
| 77 | **0.490** | **0.821692** | **0.549171** | **0.272** | **5** | **2** | **5** | **3** | **15** |
| 78 | **0.921** | **0.583694** | **0.893743** | **0.880** | **1** | **5** | **2** | **2** | **10** |
| 79 | **0.280** | **0.952888** | **0.810686** | **0.562** | **3** | **1** | **2** | **5** | **11** |
| 80 | **0.263** | **0.434511** | **0.174180** | **0.102** | **3** | **5** | **2** | **2** | **12** |
| 81 | **0.603** | **0.366292** | **0.630508** | **0.195** | **4** | **4** | **4** | **2** | **14** |
| 82 | **0.292** | **0.802139** | **0.227630** | **0.282** | **3** | **2** | **3** | **3** | **11** |
| 83 | **0.255** | **0.965725** | **0.721741** | **0.098** | **3** | **1** | **3** | **1** | **8** |
| 84 | **0.349** | **0.296367** | **0.594027** | **0.253** | **4** | **3** | **5** | **3** | **15** |
| 85 | **0.010** | **0.722206** | **0.507453** | **0.248** | **1** | **3** | **5** | **3** | **12** |
| 86 | **0.876** | **0.344469** | **0.337087** | **0.930** | **2** | **4** | **4** | **1** | **11** |
| 87 | **0.087** | **0.564958** | **0.651273** | **0.697** | **1** | **5** | **4** | **4** | **14** |
| 88 | **0.023** | **0.432682** | **0.893509** | **0.247** | **1** | **5** | **2** | **3** | **11** |
| 89 | **0.798** | **0.850869** | **0.599588** | **0.371** | **3** | **2** | **5** | **4** | **14** |
| 90 | **0.551** | **0.542743** | **0.854390** | **0.456** | **5** | **5** | **2** | **5** | **17** |
| 91 | **0.743** | **0.386456** | **0.273376** | **0.766** | **3** | **4** | **3** | **3** | **13** |
| 92 | **0.870** | **0.580082** | **0.562660** | **0.115** | **2** | **5** | **5** | **2** | **14** |
| 93 | **0.439** | **0.948720** | **0.210588** | **0.207** | **5** | **1** | **3** | **3** | **12** |
| 94 | **0.276** | **0.125789** | **0.618151** | **0.506** | **3** | **2** | **4** | **5** | **14** |
| 95 | **0.669** | **0.763418** | **0.839908** | **0.464** | **4** | **3** | **2** | **5** | **14** |
| 96 | **0.689** | **0.528420** | **0.903029** | **0.664** | **4** | **5** | **1** | **4** | **14** |
| 97 | **0.700** | **0.704109** | **0.268164** | **0.128** | **3** | **3** | **3** | **2** | **11** |
| 98 | **0.075** | **0.673992** | **0.418701** | **0.135** | **1** | **4** | **5** | **2** | **12** |
| 99 | **0.193** | **0.584235** | **0.151289** | **0.480** | **2** | **5** | **2** | **5** | **14** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 100 | **0.064** | **0.421007** | **0.324078** | **0.398** | **1** | **5** | **4** | **4** | **14** |
| 101 | **0.173** | **0.514648** | **0.301427** | **0.245** | **2** | **5** | **4** | **3** | **14** |
| 102 | **0.698** | **0.605551** | **0.415843** | **0.975** | **4** | **4** | **5** | **1** | **14** |
| 103 | **0.694** | **0.536050** | **0.790326** | **0.731** | **4** | **5** | **3** | **3** | **15** |
| 104 | **0.394** | **0.123289** | **0.317613** | **0.615** | **4** | **2** | **4** | **4** | **14** |
| 105 | **0.945** | **0.877709** | **0.296066** | **0.043** | **1** | **2** | **3** | **1** | **7** |
| 106 | **0.577** | **0.468648** | **0.303547** | **0.168** | **5** | **5** | **4** | **2** | **16** |
| **107** | **0.151** | **0.752718** | **0.383326** | **0.017** | **2** | **3** | **4** | **1** | **10** |