Tesis de Licenciatura
Secuencias Maximizadoras de Subcadenas
¿Cuán aleatorias son?

Alejandro Dau

ad1n@dc.uba.ar

Directora: Dra. Verónica Becher

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

2 de junio de 2004

Resumen

Las Secuencias Maximizadoras de Subcadenas (SMS) son una familia de secuencias infinitas que se caracterizan por maximizar la cantidad de subcadenas distintas que contienen sus prefijos. En este trabajo las presentamos mediante tres definiciones alternativas, y demostramos que son equivalentes entre sí. En base a una de las definiciones presentamos un algoritmo eficiente para generarlas. La familia de secuencias SMS incluye a dos secuencias binarias particulares estudiadas anteriormente: una fue dada por A. Ehrenfeucht y J. Mycielski al mismo tiempo que dejan la pregunta abierta de cuán aleatoria es desde el punto de vista estadístico. Y la otra fue propuesta por C. Kimberling, dejando abierta la pregunta de si posee la propiedad de disyuntividad (una secuencia es disyuntiva si todas las cadenas del alfabeto son subcadenas de la secuencia dada). En esta tesis respondemos estas preguntas de manera general. Demostramos que toda secuencia de la familia de las SMS (para cualquier alfabeto) es disyuntiva. Estudiamos estadísticamente algunas SMS, analizando las semejanzas y diferencias que se observan al compararlas con una secuencia normal, la secuencia de Champernowne, y con una de ruido aleatorio. Concluimos que las SMS no son una buena fuente de aleatoriedad.

Índice

1. Introducción **2**

2. Definiciones preliminares **3**

3. Presentación y definición de las SMS **4**

4. Definiciones equivalentes **4**

5. Las SMS son disyuntivas **7**

6. Un algoritmo para SMSs **9**
6.1. Complejidad tiempo y espacio del algoritmo 10

7. Test estadísticos sobre una SMS en base 2 **11**
7.1. Muestras utilizadas en los experimentos 11
7.2. Patrones de bits y compresibilidad estadística de n 11
7.3. ¿Es n una cadena pseudoaleatoria? 16
7.3.1. χ^2 16
7.3.2. Test de permutaciones 18
7.4. Convergencia al límite de normalidad de n 20
7.4.1. Aparición de nuevas palabras 23
7.5. Patrones en SMSs de bases mayores a 2 24

8. Posdata: Antecedentes y conclusiones **27**

Bibliografía **29**

9. Apéndices **30**
9.1. Algunas SMSs en la Encyclopedia of Integer Sequences . . . 30
9.2. Copia de nuestra demostración de disyuntividad publicada en Crux Mathematicorum 30
9.3. Listados 37
9.3.1. Generador de SMS - subsubMultibase.c 37
9.3.2. permut.pl - Programa perl usado para testear aleatoriedad mediante el método de permutaciones (Knuth [10]) 42

1. Introducción

En este trabajo presentaremos lo que denominamos "secuencias maximizadoras de subcadenas", o SMS. Damos tres definiciones equivalentes y en base a una de estas definiciones damos un algoritmo que las genera. Demostramos que las SMS contienen a todas las cadenas (a esta propiedad se la conoce como *disyuntividad* o *w-densidad*). Realizamos una serie de tests estadísticos sobre una SMS generada en el alfabeto $\{0, 1\}$ y los comparamos con los de la conocida secuencia de Champernowne [4] y con una secuencia de propiedades de ruido blanco. El resultado de estos análisis estadísticos muestra que nuestra SMS :

1. no admite compresibilidad estadística y es indistinguible al compararla gráficamente con ruido aleatorio;
2. pasa el test de χ^2 si se consideran tamaños de palabra cortos;
3. no pasa el test de permutaciones;
4. la velocidad con que aparecen todas las subcadenas en nuestra SMS es mucho mayor que la que tiene la secuencia de Champernowne, y también mayor a la que tiene el ruido aleatorio.

Una vez finalizada la investigación que exponemos en este trabajo encontramos un antecedente de la SMS en el alfabeto $\{0, 1\}$ (que utilizamos para nuestros análisis estadísticos) en [9], donde justamente enunciaba el problema de si esta secuencia tiene la propiedad de ser disyuntiva. Nuestra solución a este problema, formulado y abierto desde 1997, fue publicada en [6].

2. Definiciones preliminares

A continuación damos algunas definiciones de términos utilizados en nuestra discusión. Siendo Σ un conjunto finito de símbolos, llamamos *cadena* a una sucesión finita de símbolos de Σ yuxtapuestos. Representamos a la cadena vacía con λ y escribimos $c = c_0...c_n$, $c_i \in \Sigma$, $0 \leq i \leq n$, para la cadena c cuya longitud $|c|$ es $n+1$. Llamamos *secuencia* a una yuxtaposición de símbolos infinita a derecha $s = s_0 s_1...$, $s_i \in \Sigma$. Usamos Σ^* y Σ^ω para el conjunto de todas las cadenas y secuencias en el alfabeto Σ, respectivamente. En ocasiones llamaremos *dígitos* a los símbolos que componen una cadena o secuencia. Asumimos que Σ tiene al menos dos símbolos, y utilizamos $\#(\Sigma) = k$ para denotar su cantidad de elementos. Decimos que cadena (o secuencia) $c \in \Sigma^*$ ($s \in \Sigma^\omega$) está en *base* k, si $\#(\Sigma) = k$. Una cadena b se dice *subcadena* de una secuencia s (o cadena c) si existe una cadena a y una secuencia t (cadena d) tal que $s = abt$ ($c = abd$). La cadena a es prefijo de la cadena $c \in \Sigma^*$ (o la secuencia $s \in \Sigma^\omega$) si a es una subcadena inicial de c (s). Recíprocamente a es sufijo de la cadena c si a es subcadena final de c.

Definimos la función $P_{\Sigma^*} : \Sigma^* \to 2^{\Sigma^*}$ que da el conjunto de subcadenas de una cadena dada, es decir, $P_{\Sigma^*}(c \in \Sigma^*) := \{d \in \Sigma^* : \exists a, t \in \Sigma^*, c = adt\}$.

La función $\#_{\Sigma^*} : \Sigma^* \to I\!N$ da la cantidad de subcadenas distintas de una cadena, es decir $\#_{\Sigma^*}(c) = \#(P_{\Sigma^*}(c))$. Es facil demostrar que para $c \in \Sigma^*$ en base b,

$$|c| \leq \#_{\Sigma^*}(c) \leq \sum_{l=1}^{|c|} min(b^l, |c| + 1 - l) \leq \frac{|c|(|c|+1)}{2}$$

$\#_{\Sigma^*}(c) = |c|$ cuando c está compuesta por $|c|$ dígitos iguales. $\#_{\Sigma^*}(c)$ es igual a $\frac{|c|(|c|+1)}{2}$ sólo cuando está compuesta por $|c|$ símbolos distintos. Si para algun largo l la cadena c contiene a todas las cadenas de largo l, y además c no contiene repeticiones de ninguna cadena de largo $l+1$, $\#_{\Sigma^*}(c) = \sum_{l=1}^{|c|} min(b^l, |c| + 1 - l)$

Dadas $c, d \in \Sigma^*, |c| = |d|$, decimos que c es *más compacta* que d si $\#_{\Sigma^*}(c) \geq \#_{\Sigma^*}(d)$. Por ejemplo, la cadena 0100011 es más compacta que la cadena 0110110, ya que la primera contiene $2+4+5+4+3+2+1 = 21$ subcadenas distintas, mientras que la segunda contiene 2+3+3+3+3+2+1 = 17 subcadenas distintas.

3. Presentación y definición de las SMS

Definición 1 (SMS) *Llamamos a una secuencia* $s \in \Sigma^\omega, s = s_0 s_1 ...$ maximizadora de subcadenas *(o abreviando SMS) cuando, para cada* $i \in \mathbb{N}$, $\#_{\Sigma^*}(s_0 ... s_{i-1} s_i) \geq \#_{\Sigma^*}(s_0 ... s_{i-1}\sigma)$, *para todo* $\sigma \in \Sigma$. *Es decir,* $s \in \Sigma^\omega$ *es SMS si y solo si cumple:*

- $s_0 \in \Sigma$
- $s_{i+1} \in \{\sigma \in \Sigma : \#_{\Sigma^*}(s_0 ... s_i \sigma) \geq \#_{\Sigma^*}(s_0 ... s_i \sigma') \forall \sigma' \in \Sigma \}$

Informalmente, el algoritmo que construye estas secuencias tiene por objetivo hacer que cada prefijo de las mismas contenga una cantidad máxima de subcadenas distintas: la secuencia se genera extendiendo de a un dígito su cadena prefijo, tal que ese dígito sea el que maximiza la cantidad de subcadenas distintas.

Nuestro algoritmo recuerda al método de permutaciones propuesto por M. H. Martin [12], al que hace referencia Knuth en [10]. Mediante el método de Martin es posible construir una cadena de largo $m^l + l - 1$, que contiene las m^l posibles cadenas de largo l en base m, sin repeticiones. Estas cadenas también se conocen como *Cadenas de De Bruijn* [7]. Este método aprovecha al máximo la longitud de la cadena: la cadena final contiene una cantidad máxima de subcadenas de largo l; sin embargo este método solo es aplicable si se desea concatenar una cantidad finita de dígitos, conocida de antemano. Sus prefijos no son maximales en la cantidad de subcadenas que presentan, por lo que no es útil si lo que se busca es construir una secuencia (cadena infinita) o una cadena de largo a priori indefinido.

4. Definiciones equivalentes

La definición 1 presenta a las SMS en base al criterio de maximizar la cantidad de subcadenas distintas en sus prefijos. A continuación damos dos definiciones alternativas y demostramos que las tres son equivalentes entre sí.

Definición por mínimo largo de nueva subcadena

Definición 2 (SMLNS) *Sea la función* $\not{t}(q, c)$ *que da el sufijo de largo* q *de la cadena* c *y* $P_{\Sigma^*}(c)$ *que da el conjunto de subcadenas que aparecen en una cadena* c. *Llamamos SMLNS a la secuencia* s *si:*

- $s_0 \in \Sigma$
- $s_{i+1} \in \{\sigma \in \Sigma / \forall \sigma' \in \Sigma, \quad \min_q (\not{t}(q, s_0 ... s_i \sigma) \notin P_{\Sigma^*}(s_0 ... s_i)) \leq \min_q (\not{t}(q, s_0 ... s_i \sigma') \notin P_{\Sigma^*}(s_0 ... s_i)) \quad \}$

Proposición 3 *Sea $s \in \Sigma^\omega$. s es SMLNS si y solo si s es SMS.*

DEMOSTRACIÓN. Llamamos "nueva subcadena de $s_0...s_n$" a una cadena que aparece en $s_0...s_n$ pero no en $s_0...s_{n-1}$. Al extender en un símbolo una cadena de largo n, la cadena resultante tiene al menos una cadena nueva, la misma de largo $n+1$; es decir, $\#_{\Sigma^*}(s_0...s_{n-1}s_n) = \#_{\Sigma^*}(s_0...s_{n-1}) + k$, para algún $k \geq 1$. Todas las subcadenas nuevas tienen que ser sufijo de $s_0...s_{n-1}s_n$, ya que de lo contrario no serían nuevas, es decir ya estarían en $s_0...s_{n-1}$. Como hay a lo sumo $n+1$ subcadenas con tal sufijo, $1 \leq k \leq n+1$. Sea $s_{n-q}...s_n$ la subcadena nueva de largo menor. Entonces todas las subcadenas que la extienden también serán nuevas. Por lo tanto, $k = n - q + 1$: la cantidad de subcadenas distintas se maximiza cuando el largo de la menor subcadena nueva se minimiza, y viceversa. Es decir, una secuencia es una SMS sii es una SMLNS. ⊗

Definición por primeras apariciones de c en s

Definición 4 (SC) *Llamamos a una secuencia $s \in \Sigma^\omega$ una Secuencia Completa o SC si para todo $c \in \Sigma^*$, las primeras $\#(\Sigma)$ apariciones de c en s preceden a símbolos distintos. Es decir*

$$\forall i, j, 1 \leq i < j \leq \#(\Sigma), s_{I(s,c,i)+|c|} \neq s_{I(s,c,j)+|c|}$$

donde $I(s,c,i)$ es el índice de la i-ésima aparición de c en s ($i \geq 1$).

Proposición 5 *Sea $s \in \Sigma^\omega$. s es SMS si y solo si s es SC.*

DEMOSTRACIÓN.

($\Rightarrow$). Supongamos que $s \in \Sigma^\omega$ es SMS pero no SC. Como s no es SC, existe una cadena $c \in \Sigma^*$ y un $\tau \in \Sigma$ tales que

$$\exists i, j, 1 \leq i < j \leq \#(\Sigma), s_{I(s,c,i)+|c|} = s_{I(s,c,j)+|c|} = \tau$$

es decir, dos de las primeras $\#(\Sigma)$ apariciones de c son sucedidas por el mismo símbolo τ. Ahora bien, por el criterio anterior y como s es SMS, $s_{I(s,c,j)+|c|}$ es tal que minimiza el largo de la menor cadena nueva de $s_0...s_{I(s,c,j)}...s_{I(s,c,j)+|c|}$. Por hipótesis, $c\tau$ ya aparecía en $s_0...s_{I(s,c,j)+|c|-1}$, por lo que la cadena nueva de largo menor al agregarle τ será de largo mayor a $|c\tau|$. Como $0 < j < \#(\Sigma)$, tiene que existir un $\tau' \in \Sigma$ tal que $\forall i, 1 \leq i \leq j, \tau' \neq s_{I(s,c,i)+|c|}$. Por lo tanto, $c\tau'$ no aparecía en $s_0...s_{I(s,c,j)+|c|-1}$, entonces usar τ' en vez de τ en la posición $I(s,c,j)+|c|$ hubiese dado una cadena nueva de largo menor o igual a $|c\tau|$. Esto contradice la hipótesis de que s es SMS.

$\Leftarrow$). Supongamos que s es SC pero no SMS. Entonces existe un índice $i+1$ tal que s_{i+1} no minimiza el largo de la subcadena nueva menor, es decir:

$$s_{i+1} \notin \{\sigma \in \Sigma / \quad \text{mín}_q\,(\mathbb{t}(q, s_0...s_i\sigma) \notin P_{\Sigma^*}(s_0...s_i)) \leq \text{mín}_q\,(\mathbb{t}(q, s_0...s_i\sigma') \notin P_{\Sigma^*}(s_0...s_i))\}$$

O sea,

$$\exists\sigma \in \Sigma / \quad \text{mín}_q\,(\mathbb{t}(q, s_0...s_i\sigma) \notin P_{\Sigma^*}(s_0...s_i)) < \text{mín}_q\,(\mathbb{t}(q, s_0...s_is_{i+1}) \notin P_{\Sigma^*}(s_0...s_i)) \quad (*)$$

es decir, el menor sufijo de $s_0...s_i\sigma$ que no está en $s_0...s_i$, es de largo menor que el menor sufijo de $s_0...s_is_{i+1}$ que no está en $s_0...s_i$.

Tomemos el menor de tales $i+1$; para todo $s_j, j < i+1$ se cumplirán las restricciones de las SMS. Sea $p = s_0...s_i$.
Como s es SC, las primeras $b = \#(\Sigma)$ apariciones de cualquier cadena c en s preceden a símbolos distintos. Para todo sufijo t de p puede darse uno de estos tres casos:

A. t aparece menos de b veces en p
B. t aparece exactamente b veces en p
C. t aparece más de b veces en p

En particular tomemos como t el sufijo de p de mínimo largo tal que t cumple A o B (siempre existe alguno, en el peor caso $p = t$).
Si t cumple B, y como s es SC, las $b-1$ anteriores apariciones de t en p tienen como símbolo sucesor a los $b-1$ símbolos de Σ distintos de s_{i+1}. Entonces, ts_{i+1} es una cadena nueva, y haber puesto cualquier símbolo $\sigma \neq s_{i+1}$ hubiera dado una cadena nueva de largo mayor. Pero esto contradice (*).
Si t cumple A, las $k < b$ veces que aparece t en p tienen como símbolo sucesor a símbolos distintos en Σ, y como s es SC, s_{i+1} también será un símbolo distinto. Entonces, ts_{i+1} es una cadena nueva. Por hipótesis, todo sufijo propio t' de t, con largo $q < k$, estará en la categoría C) (porque elegimos t de mínimo largo cumpliendo A o B), por lo tanto $\forall\sigma \in \Sigma, t'\sigma$ no será cadena nueva. Luego ts_{i+1} es una cadena nueva de mínimo largo, lo que contradice (*). $\otimes$

Como corolario inmediato notemos que, si s es una SMS, $s_0..s_{b-1}$ está compuesto por una permutación de los b símbolos en Σ.

5. Las SMS son disyuntivas

La propiedad de disyuntividad de una secuencia es también conocida como la propiedad de ω-densidad y significa que una secuencia del alfabeto Σ contiene en su expansión infinita a todas las palabras de Σ^*.

Definición 6 *Una secuencia $s \in \Sigma^\omega$ es* disyuntiva *si, para todo $w \in \Sigma^*$, w aparece en s, es decir, existe $i \in \mathbb{N}$ tal que $s_i \dots s_{i+|w|} = w_1 \dots w_{|s|}$.*

El siguiente teorema implica que las SMS son disyuntivas.

Teorema 7 *Si $s \in \Sigma^\omega$ es una SMS entonces toda cadena $w \in \Sigma^*$ aparece infinitas veces en s.*

DEMOSTRACIÓN. Sea $s \in \Sigma^\omega$ una SMS. Es trivial ver que la cadena λ de largo 0 aparece infinitas veces en s. Veamos que $w \in \Sigma^*, w = d_1 ... d_n$ aparece infinitas veces, si $w^- \in \Sigma^*, w^- = d_1 ... d_{n-1}$ aparece infinitas veces. Supongamos que existe un $w \in \Sigma^*, |w| \geq 1$, que aparece en s solo una cantidad finita k de veces, $k \geq 0$. Por hipótesis, el prefijo de w, $w^- = d_1 ... d_{n-1}$, aparece infinitas veces en s. Supongamos que $k = 0$, es decir que w no aparece en s. Pero dado que w^- aparece infinitas veces, las primeras $b = \#(\Sigma)$ veces está sucedida por símbolos distintos, en particular uno de esos símbolos debe ser d_n, haciendo aparecer $d_1 ... d_{n-1} d_n = w$, absurdo.

Sea entonces $k \geq 1$, es decir que w aparece al menos una vez en s. Supongamos que la última aparición de w finaliza en la posición p. De esta manera, $s_{p-n+1} ... s_p = w$, y esta es la última aparición de w en s. En consecuencia, no hay cadenas de longitud $p+2$ que tengan a w como sufijo. Sin embargo, necesariamente hay alguna cadena de longitud $p+1$ que tenga como sufijo a w^- y aparezca en s infinidad de veces, ya que w^- aparece infinidad de veces y solo hay una cantidad finita de cadenas en Σ^* de largo p. En particular, las primeras b apariciones de esa cadena deberán estar sucedidas por b símbolos distintos, entre los que debería estar d_n, absurdo. El absurdo viene de suponer que las SMS no son disyuntivas, es decir, que hay palabras que aparecen solo una cantidad finita de veces en una SMS.

$\otimes$

Conjetura: las SMS son secuencias normales

En la búsqueda de una definición de aleatoriedad, en 1909 Emil Borel definió el concepto de normalidad para secuencias infinitas: una secuencia $s \in \Sigma^\omega$ es normal si todas las cadenas $w \in \Sigma^*$ ocurren con frecuencia $1/b^{|w|}$ en la expansión infinita de s, donde $b = \#(\Sigma)$, $b \geq 2$.

Definición 8 (Borel [2]) *$s \in \Sigma^\omega$ es normal en base b si y solo si*

$$\forall w \in \Sigma^* \quad \lim_{n\to\infty} \frac{\textit{cantidad de apariciones de } w \textit{ en } s_0 s_1 ... s_{n-1}}{n} = \frac{1}{b^{|w|}}$$

Es decir, no solamente s incluye a todas las cadenas, sino que éstas aparecen con igual frecuencia relativa.

La definición de Borel se extiende inmediatamente a números reales considerando su expansión decimal en una base dada. Borel luego define el concepto *absoluta normalidad*: un real es absolutamente normal cuando es normal en toda base mayor o igual que 2. En el mismo trabajo [2] Borel demuestra que el conjunto de números absolutamente normales tiene medida de Lebesgue 1, lo que significa que con probabilidad 1 un número real escogido al azar es absolutamente normal. Hasta el día de hoy persiste la conjetura de que las constantes irracionales fundamentales de las matemáticas como π, e [1] son absolutamente normales.

Champernowne [4] construyó el primer ejemplo de secuencia normal en base 10:

Champernowne 1234567891011121314....

En [5] se generalizó el método usado por Champernowne demostrando que la secuencia de dígitos de números primos 23571113171923... también es normal en base 10.

La propiedad de disyuntividad de una secuencia es condición necesaria para la de normalidad, pero no suficiente. Conjeturamos que las SMS son normales en la base en la que fueron definidas. En la seccion 7.4 mostraremos resultados estadísticos que soportan nuestra conjetura.

6. Un algoritmo para SMSs

En esta sección discutimos algunas cuestiones acerca de la implementación de un algoritmo que genera SMSs.

Función de decisión

De la definición 1 de las SMSs se ve que cada símbolo que la compone es elegido de un subconjunto de Σ. Para implementar la generación de una SMS, definiremos una función de decisión $d : \mathbb{N} \times 2^{\Sigma} \rightarrow \Sigma$, que dado un natural y un subconjunto no vacío de Σ, da un elemento de ese subconjunto. Esa función será usada por el algoritmo para decidir qué símbolo de los posibles es el siguiente en la secuencia, cada vez que haya símbolos que empatan (con respecto a la cantidad de subcadenas diferentes que inducen en el prefijo extendido).

Algoritmo

El siguiente es un algoritmo generador de SMSs, escrito en pseudocódigo. Se utilizan las fuciones *sufijo(l, S)* , que da la cadena sufijo de largo l de S, y *len(S)* que da la cantidad de símbolos que componen la cadena S.

Parametros[1] del algoritmo:

Σ	Base en la que se generara la secuencia (conjunto de simbolos).
`L`	Maximo largo de sufijo a considerar.
`d(int,set)`	Funcion de decision. Devuelve uno de los elementos del segundo parametro. En cada invocacion de la funcion el primer parametro tendra un valor distinto, lo que permitira generar cualquier SMS. (podemos no querer que la desicion sea siempre la misma ante un mismo conjunto de alternativas)

Algoritmo:

```
S=λ;                                  comienzo con la cadena vacia
Cadenas = { λ };                      por ahora la unica subcadena de S es λ.

Mientras (true) {
    posibles = {};                    inicializo conjunto de simbolos elegibles
    l = 0;                            empiezo por considerar el largo de sufijo menor
    Mientras (posibles == {}) {
        l = l+1;                      incremento el largo de sufijo a considerar
        Si (l > L) END;               si excedo el limite L termino la ejecucion
```

[1] Por problemas de formateo de mi LaTeX debí evitar usar acentos en la descripción del algoritmo

```
        ParaCada( s ∈ Σ ) {              busco posibles extensiones de S
            Si ( sufijo( l, S++s ) ∉ Cadenas )
                posibles = posibles ⋃ { s };
        }
    }                                    aqui posibles tiene las posibles extenciones de S
    S = S ++ d( len(S), posibles );      decido entre los posibles y extiendo S
    ParaCada ( 1 ≤ i ≤ L ) {             agrego a Cadenas los L sufijos de S mas cortos
        Cadenas = Cadenas ⋃ { sufijo( i, S ) };
    }
}
```

El algoritmo termina cuando alcanza la cota del largo de sufijo que es necesario considerar para conseguir el siguiente símbolo. Esta cota determina la cantidad de símbolos que se van a obtener. Vale la pena comentar que a partir de los resultados de nuestras pruebas en distintos alfabetos Σ, conjeturamos que la cantidad de dígitos obtenidos considerando sufijos de tamaño máximo L es del orden de b^L. No tenemos una demostración que confirme esta conjetura.

En el apéndice 1 se muestra una implementación en lenguaje C de este algoritmo. Permite obtener los primeros dígitos de las SMS con $\Sigma = \{1, \ldots, b\}$ con $1 < b < 10$, y función de decisión $d(n, S) = min(S)$. El parámetro L dependerá de la cantidad de memoria que se desee utilizar, ya que el conjunto *Cadenas* está implementado usando un byte por elemento posible (es decir, la memoria usada es proporcional a b^L).

6.1. Complejidad tiempo y espacio del algoritmo

En cada paso este algoritmo calcula los símbolos que, agregados a S, hacen nuevas cadenas de largo mínimo. El orden de complejidad de este algoritmo depende de la cantidad de símbolos de Σ y la cota al tamaño máximo de prefijo a considerar L. La complejidad de agregar un dígito a un prefijo de largo i es del orden $O(i * b)$ donde b es la cantidad de símbolos de Σ, y en el peor de los casos i es igual al tamaño máximo del prefijo a considerar L. La estructura de datos para el conjunto *Cadenas* tiene $O(1)$ para chequear la existencia de un elemento. Y la cantidad de espacio para el almacenamiento de las marcas para almacenar este conjunto, es $O(b^{L+1})$.

7. Test estadísticos sobre una SMS en base 2

En esta sección mostramos algunos resultados obtenidos al comparar una SMS en base 2 con otras dos secuencias significativas. Como ya lo hemos mencionado en la Introducción, luego de haber generado nuestra secuencia hemos encontrado una referencia a ella en [9], problema de Kimberling en Crux Mathematicorum (ver Apéndice).

7.1. Muestras utilizadas en los experimentos

Tomaremos los primeros 11,000,000 (once millones) de símbolos de las siguientes secuencias:

- SMS en base 2, con función de decisión $d(i, Posibles) = min(Posibles)$. Vale la pena notar que esta secuencia tiene la particularidad de ser la menor en orden lexicográfico de las SMS de base dos. (la menor base posible)
  Llamaremos n al prefijo de esta secuencia que usaremos para las pruebas.

- Secuencia de Champernowne [4] en base 2. Esta secuencia esta compuesta por la concatenación de los dígitos significativos de los números naturales expresados en base dos. Esta secuencia es normal en base 2. Llamaremos *champ* al prefijo de esta secuencia que usaremos para las pruebas.

- Bits del archivo *bits.01* del "Marsaglia Random Number CDROM" [11]. Los bits de este archivo fueron obtenidos mediante procedimientos físicos generadores de ruido aleatorio, y pasan satisfactoriamente el conjunto de pruebas del mismo autor "The Diehard Battery of Tests of Randomness". Llamaremos *rand* al prefijo de esta cadena que usaremos para las pruebas.

En el Cuadro 1, mostramos los primeros 180 símbolos de *champ*, n y *rand*.

7.2. Patrones de bits y compresibilidad estadística de n

La Figura 2 contiene el primer megabit de *champ*. Tomando el punto $(0, 0)$ como el del extremo superior izquierdo de la figura, el punto (x, y) es negro sii la posición $y * 1024 + x$ de la cadena *champ* es 1. En este gráfico se manifiesta el "método de generación" de la secuencia de Champernowne: simplemente concatenar los dígitos de números naturales correlativos. Al ver miles de bits de esta secuencia, es fácil para un humano reconocer patrones: líneas o regiones en la figura donde es mucho más probable encontrar un dígito que otro, o subcadenas que se repiten periódicamente con pequeñas variaciones.

champ	1 1 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1 1 0 0 0 1 0 0 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 1 0 0 1 1 1 0 1 0 0 1 0 1 0 1 1 0 1 1 0 1 0 1 1 1 1 1 0 0 0 1 1 0 0 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 0 0 0 1 1 1 0 0 1 0 0 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1 1 1 0 1
n	0 1 0 0 0 1 1 0 1 0 1 1 1 0 0 1 0 0 1 1 1 1 0 1 1 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 0 1 0 0 1 0 1 1 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 1 1 1 1 1 1 0 1 0 1 0 1 0 0 1 1 0 1 1 0 1 0 0 0 1 0 1 1 1 1 0 0 1 1 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 1 0 0 1 0 1 0 1 1 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 0 0 1 0 1 1 1 0 1 1 1 0 0 0 1 1 0 0 0 0
rand	1 1 0 0 0 0 0 1 1 0 0 1 0 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 0 1 0 1 0 0 0 1 0 0 1 0 1 1 1 0 1 1 1 0 1 0 0 0 1 1 1 1 0 0 1 1 0 1 1 0 0 0 1 0 1 1 1 1 0 1 0 0 1 0 0 0 1 1 1 0 0 1 1 1 1 0 0 0 0 1 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 1 0 1 0 0 1 0 1 0 0 0 1 1 0 0 1 0 0 1 1 1 0 1 0 1 1 0 0 1 0 1 0 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0

Cuadro 1: Los primeros 180 dígitos de las muestras *champ*, *n* y *rand*.

La muestra de la SMS en base 2, n, también se construye con reglas simples:

a. Si es lo mismo poner 0 que 1, poner 0.

b. Toda cadena es sucedida por símbolos distintos en sus dos primeras apariciones.

La regla *a* sugiere que se privilegiará a los 0 en la composición de la cadena. ¿Provocará esto que la proporción de símbolos 1 sea menor que la de 0? Si la regla *b* hace que las proporciones se balanceen, ¿la proporción de 0s al principio de la cadena será mayor? Más en general, ¿se podrá observar fácilmente el patrón de generación de la cadena si graficamos algunos miles de términos?

La Figura 3 contiene el primer megabit de esta SMS, con el mísmo método usado en la Figura 2. A simple vista no se observa ningún tipo de patrones regulares, cómo los apreciables en el gráfico de Champernowne. La figura de n recuerda al dibujo que hace el ruido blanco sobre una pantalla de televisión: cuando se sintoniza un aparato de televisión a una frecuencia donde no es transmitida una señal, se dibujan en la pantalla "secuencias aleatorias" que están en las ondas de radio generadas por la naturaleza.

Para la Figura 4 usamos los dígitos de *rand*, publicados en [11]. Estos dígitos fueron obtenidos por métodos no determinísticos, y pasaron todas las pruebas de aleatoriedad de Marsaglia. Simplemente mirando las figuras 4 y 3, no se observan características que permitan distinguirlas. Ambos gráficos

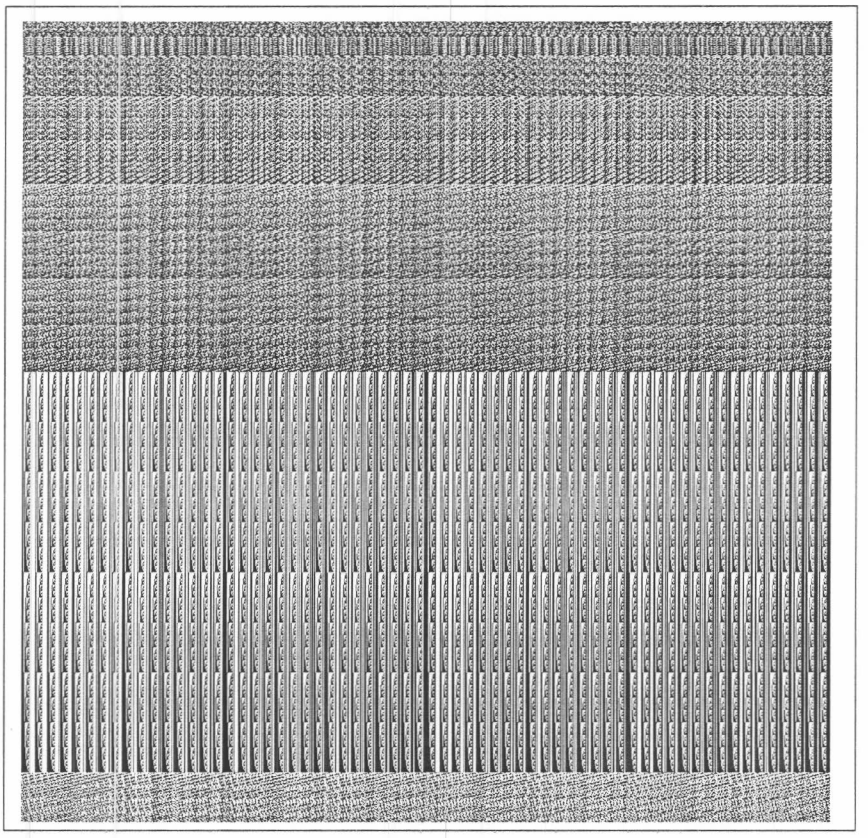


Figura 2: Los 1024^2 primeros bits de *champ* (prefijo de la secuencia de Champernowne en base 2)

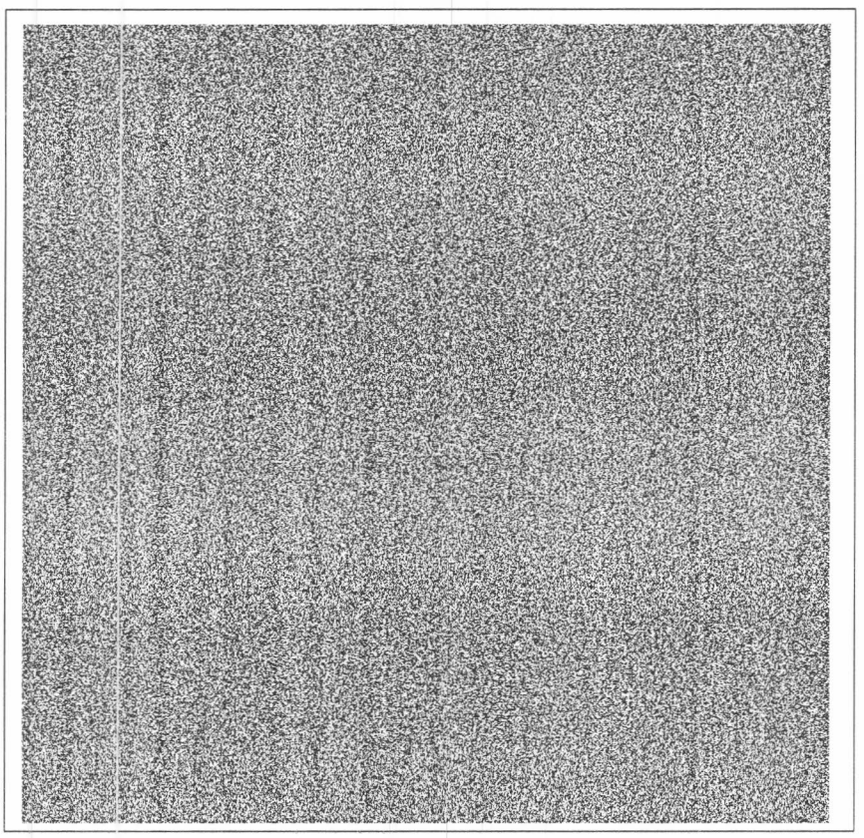


Figura 3: Los 1024^2 primeros bits de n (SMS en base 2 con función de decisión $f(i, Posibles) = min(Posibles)$)

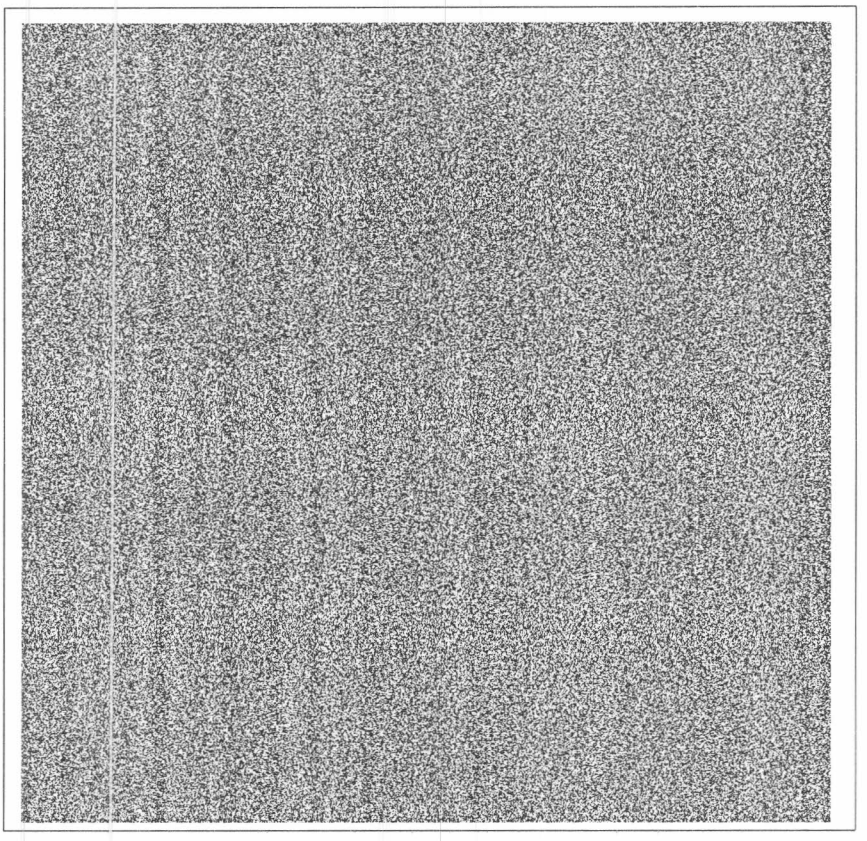


Figura 4: Los 1024^2 primeros bits de *rand* (cadena aleatoria de Marsaglia [11])

tienen una distribución uniforme de sus puntos, sin patrones repetitivos.

La eficiencia de los algoritmos de compresión de datos está dada en buena medida por la capacidad de reconocer patrones en su entrada. A partir de los datos de entrada estos algoritmos generan una cadena de salida, de tamaño menor si es que detectan patrones o redundancia que pueda ser expresada con menos cantidad de símbolos. Esa cadena de salida tiene toda la información necesaria para reconstruir la cadena original usando un algoritmo de descompresión establecido.

En la Figura 5 se grafica el resultado de comprimir las muestras mediante los algoritmos Lempel-Ziv 77 (implementado en la utilidad gzip) y Burrows-Wheeler (implementado en la utilidad bzip2). En la única muestra en la que los algoritmos encontraron redundancia fue en *champ*. Tanto en *rand* como en n, el resultado de las compresiones fue de tamaño mayor a la de entrada. Es decir, no encontraron una cadena de tamaño menor o igual a

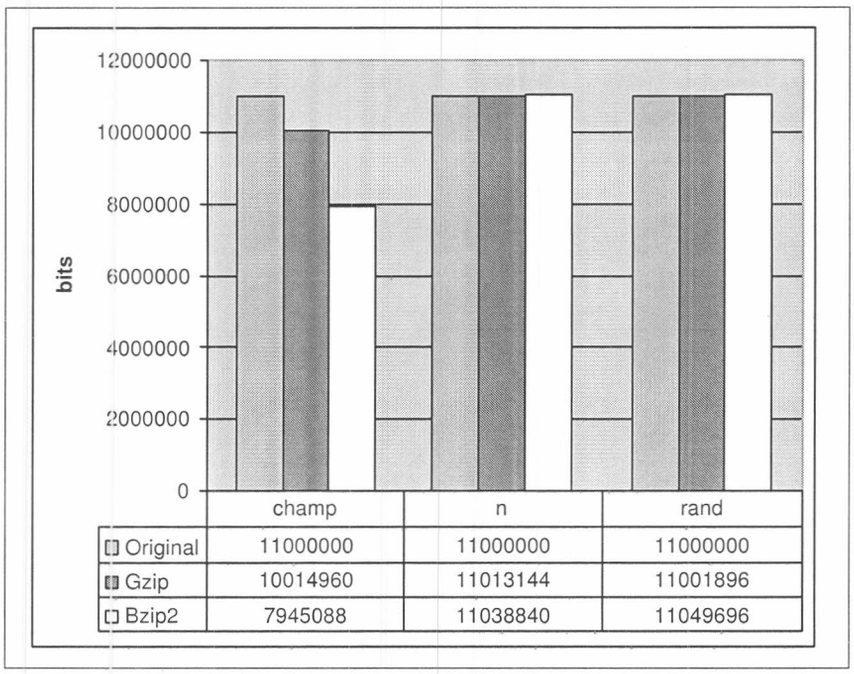


Figura 5: Tamaño de las muestras al comprimirlas usando Gzip y Bzip2.

la de entrada, que sirviese al algoritmo de descompresión para generar la cadena original.

7.3. ¿Es n una cadena pseudoaleatoria?

Ya vimos que las cadenas n y *rand* se comportan de manera similar al graficarlas y al comprimirlas. Obviamente n no es aleatoria desde un punto de vista algoritmico: no necesita de un algoritmo de tamaño similar para ser generada (ver Chaitin [3]). ¿Pero es al menos "pseudoaleatoria"? Es decir, ¿pasa los tests estadísticos de aleatoriedad que sí pasan los números obtenidos mediante algoritmos generadores de números pseudoaleatorios?

7.3.1. χ^2

El test estadístico de aleatoriedad más difundido es el χ^2. Informalmente, este test mide la distancia entre la probabilidad de que se manifieste un tipo de evento dado, y la cantidad de veces que ocurrió ese evento en la muestra analizada. A partir de esa distancia y de la cantidad de tipos de eventos, el test indica cuán probable es que los resultados obtenidos de la muestra analizada correspondan a los de una muestra "aleatoria".

		n		*rand*		*champ*	
$\|w\|$	Palabras	χ^2	P	χ^2	P	χ^2	P
1	11000000	0.3240	0.5692	0.073964	0.7857	11727.1617	0
2	5500000	2.8573	0.4141	1.7548	0.6248	22727.8078	0
3	3666666	5.1384	0.6431	3.9880	0.7812	14857.0017	0
4	2750000	15.8770	0.3903	17.3013	0.3012	69169.2878	0
5	2200000	23.6983	0.8227	19.3853	0.9481	38993.9689	0
6	1833333	51.4350	0.851	59.6552	0.5963	36657.3770	0
7	1571428	91.5288	0.9925	109.6994	0.8636	13154.1866	0
8	1375000	260.9553	0.3855	258.4817	0.4274	138550.0605	0
9	1222222	467.4901	0.9163	494.5072	0.6917	25633.1508	0
10	1100000	876.5076	0.9997	1060.2393	0.2038	86238.2387	0
11	1000000	1880.1213	0.9962	2037.2848	0.5563	13617.4387	0
12	916666	3950.0477	0.9467	4191.1518	0.1442	98814.0933	0

Cuadro 6: Tabla de resultados del test de χ^2 sobre las tres muestras

Si este valor numérico está en los extremos (demasiado cercano a 0 o a 1), podemos afirmar con bastante seguridad que la muestra no presenta comportamiento aleatorio. Para una explicación formal del método y su aplicación para evaluar generadores de números pseudoaleatorios, ver [10], págs 42-43.

En el Cuadro 6 vemos los resultados de calcular el test sobre las tres muestras. Se corrió el test variando el largo de palabra entre 1 y 12 bits, y calculando el χ^2 de los grados de libertad correspondientes (de 1 a 4095).

En las columnas P de los resultados de cada muestra, figura la probabilidad de que una secuencia aleatoria supere el valor de χ^2 obtenido. Por lo tanto, este test aplicado sobre una cadena de dígitos obtenidos por un algoritmo de generación de números pseudoaleatorios que se precie de tal, no debería dar valores demasiado cercanos a 0 o a 1. Como vemos, n tiene valores de χ^2 compatibles con los de una fuente aleatoria si tenemos en cuenta largos de palabra menores que 7. Pero para los largos de palabra mayores la probabilidad se vuelve demasiado cercana a 1, lo que indica que el valor de χ^2 resultó demasiado bajo (la cantidad de ocurrencias de cada palabra fue demasiado uniforme). Vale la pena comparar estos valores con los obtenidos para *rand* donde, como era previsible, las probabilidades asociadas no fueron extremas. El test muestra claramente que los dígitos de *champ* no fueron obtenidos de una buena fuente aleatoria: la probabilidad de que la misma cantidad de dígitos tomados al azar den un valor de χ^2 mayor es 0.

7.3.2. Test de permutaciones

Probamos ahora con un test más sofisticado: el test de permutaciones. Este test es una adaptación del presentado en Knuth [10], (pags. 65,66) para poder aplicarlo a variables aleatorias en Z_n, no en $\mathbb{R}$. Funciona de la siguiente manera. Se va tomando de la entrada tres subcadenas (adyacentes pero sin superposición) de k dígitos, y se ordenan lexicográficamente. Ese ordenamiento determina una permutación de las subcadenas originales. Además de las seis permutaciones posibles de tres elementos, se toman en cuenta los siete casos en que puede haber igualdad entre algunas de las subcadenas. Durante el proceso se lleva la cuenta de cuantas permutaciones de cada tipo han ocurrido. Finalmente se calcula un χ^2 de doce grados de libertad (por ser trece el total de posibilidades), comparando los valores esperados de ocurrencia de cada permutación, contra los obtenidos en el experimento.

El Cuadro 7 tiene los resultados del test de permutaciones sobre nuestra cadena n. Breve descripción de las columnas:

bits	largo de las palabras consideradas.
cant	cantidad de palabras consideradas.
$\{- = \equiv\}^3$	Cantidad de ocurrrencias del trío
χ^2	resultado de χ^2, para las cantidades obtenidas.
p	probabilidad asociada al resultado de χ^2 de 12 grados de libertad.

Se puede observar que para largos de palabra menores o iguales a 15 el resultado es satisfactorio: no hay valores de P extremos. Pero para la medición realizada tomando palabras consecutivas de 16 bits, el valor de χ^2 es extremadamente alto. Se puede ver que los contadores que corresponden a permutaciones con dos elementos iguales tienen valores demasiado grandes, en proporción a la cantidad de palabras y al largo de palabra considerada. Esto se refleja en el valor de χ^2, y en la probabilidad asociada.

Como conclusión, la SMS de donde se sacó n, aunque presenta algunas características de pseudoaleatoriedad, no es una buena fuente de números pseudoaleatorios.

bits	cant	− − −	− = −	− − =	− ==	= − =	= − −	== −	− =≡	− ≡=	= − ≡	≡ − =	=≡ −	≡= −	χ^2	P
1	3883008	971642	484720	485672	485669	485678	485428	484199	0	0	0	0	0	0	5.1076	0.9543
2	1941504	121610	182016	181891	182592	181694	181418	182262	121495	121155	121524	121307	120793	121747	9.9595	0.6195
3	1294336	20199	71110	71007	70972	70277	70708	71005	141214	141858	141660	141283	142071	140972	13.5464	0.3306
4	970752	3832	28643	28364	28243	28368	28280	28619	132648	132967	132663	132911	132522	132692	6.7470	0.8739
5	776601	762	11855	11676	11783	11822	11818	11902	117315	117740	117454	117453	117683	117338	5.4912	0.9395
6	647168	167	4985	4790	5088	5027	4928	4949	102828	102788	103087	102452	103049	103030	13.9806	0.302
7	554715	16	2199	2172	2079	2158	2171	2211	90568	90536	90415	90434	89900	89856	20.7838	0.0536
8	485376	6	972	937	907	910	970	929	79958	80201	79923	79840	79602	80221	8.1844	0.7706
9	431445	0	402	416	395	407	405	419	71194	71281	71679	72156	71173	71518	13.3865	0.3416
10	388300	0	172	177	184	183	207	205	64371	64524	64944	64295	64848	64190	12.9649	0.3716
11	353000	0	75	94	85	83	71	77	59181	58656	58729	58743	58687	58519	10.2169	0.5969
12	323584	0	26	41	39	47	42	43	54193	53572	53991	53866	53796	53928	10.5424	0.5685
13	298692	0	17	21	16	28	16	17	50185	49507	49857	49627	49636	49765	12.1390	0.4346
14	277357	0	6	8	6	8	10	9	46211	46546	46001	46154	46417	45981	7.3065	0.8367
15	258867	0	8	5	8	5	5	5	43157	43215	43309	43198	43083	42869	12.0776	0.4395
16	242688	0	38	50	104	34	106	144	39987	40896	40212	40391	40180	40546	24936.7093	0
17	228412	0	0	0	1	1	0	1	38235	37943	38127	37931	38207	37966	2.5654	0.9979
18	215722	0	0	0	0	1	0	2	35612	36032	36071	35691	35988	36325	16.5623	0.1668
19	204368	0	0	0	0	0	0	0	34023	33855	34404	34163	34036	33887	5.9550	0.9183
20	194150	0	0	0	0	0	0	0	32280	32308	32202	32380	32558	32422	2.3950	0.9985
21	184905	0	0	0	0	1	0	0	30838	30817	30726	30809	30916	30798	21.3425	0.0456
22	176500	0	0	0	0	0	0	0	29612	29406	29187	29261	29542	29492	4.6447	0.9688
23	168826	0	0	0	0	0	0	0	28020	28133	28204	28175	27993	28301	2.3907	0.9985
24	161792	0	0	0	0	0	0	0	26786	27126	26814	26917	27100	27049	4.0180	0.9831
25	155320	0	0	0	0	0	0	0	25710	25746	26169	25877	25848	25970	5.3790	0.9441
26	149346	0	0	0	0	0	0	0	24795	25057	24800	24865	24976	24853	2.1854	0.9991
27	143815	0	0	0	0	0	0	0	23976	24008	23761	24045	24176	23849	4.4999	0.9726
28	138678	0	0	0	0	0	0	0	23026	22957	23284	23224	23112	23075	3.2411	0.9936
29	133896	0	0	0	0	0	0	0	22398	22187	22245	22303	22328	22435	1.9215	0.9995
30	129433	0	0	0	0	0	0	0	21686	21580	21670	21389	21621	21487	3.0492	0.9952
31	125258	0	0	0	0	0	0	0	20858	20921	20971	20831	20850	20827	0.7892	1
32	121344	0	0	0	0	0	0	0	20228	20007	20371	20191	20245	20302	3.7741	0.9872
33	117666	0	0	0	0	0	0	0	19428	19676	19502	19914	19696	19450	8.9006	0.7114
34	114206	0	0	0	0	0	0	0	18945	19129	18956	19191	18954	19031	2.8416	0.9966
35	110943	0	0	0	0	0	0	0	18526	18532	18433	18319	18660	18473	3.5011	0.9909
36	107861	0	0	0	0	0	0	0	17981	17850	18096	17963	17932	18039	2.0232	0.9994
37	104946	0	0	0	0	0	0	0	17369	17435	17438	17496	17681	17527	3.3303	0.9927
38	102184	0	0	0	0	0	0	0	17026	16890	17053	17107	17092	17016	1.7681	0.9997
39	99564	0	0	0	0	0	0	0	16743	16552	16400	16620	16617	16632	3.8719	0.9856
40	97075	0	0	0	0	0	0	0	16268	15973	16133	16279	16137	16285	4.6648	0.9682

Cuadro 7: Tabla de resultados del test de permutaciones sobre n.

7.4. Convergencia al límite de normalidad de n

Recordemos que se dice que una secuencia de dígitos en base 2 $a_0a_1\ldots$ es *Normal* cuando se verifica el siguiente límite, para todo $w \in \{0,1\}^*$:

$$\lim_{i\to\infty} \frac{\#_w(a_0\ldots a_i)}{i} = 2^{-|w|}$$

donde la función $\#_w(a_0\ldots a_i)$ da la cantidad de veces que aparece la cadena w en la cadena $a_0\ldots a_i$.

Es imposible mostrar experimentalmente que la SMS que estamos estudiando cumple este límite, ya que llevaría tiempo infinito. Pero podemos ver si al menos el prefijo n estudiado se acerca a ese límite, y comparar esa progresión con las de *rand* y *champ*.

Las figuras 8 y 9 contienen el resultado del análisis de convergencia al límite de normalidad para cada muestra, y para cada largo de palabra $|w|$ entre 1 y 20. Cada 50000 dígitos se dibuja la máxima diferencia entre el límite de normalidad ($2^{-|w|}$), y la proporción de ocurrencia de las palabras del largo estudiado ($\frac{\#_w(a_0\ldots a_i)}{i}$). Esta diferencia debería acercarse a 0 a medida que se avanza en la secuencia, si la secuencia es normal.

Vemos que para todos los largos de palabra estudiados, la muestra n está a menor distancia del límite de normalidad que las otras dos muestras. En el caso de *champ* es prefijo de una secuencia normal y en el caso de *rand* la normalidad es esperada.

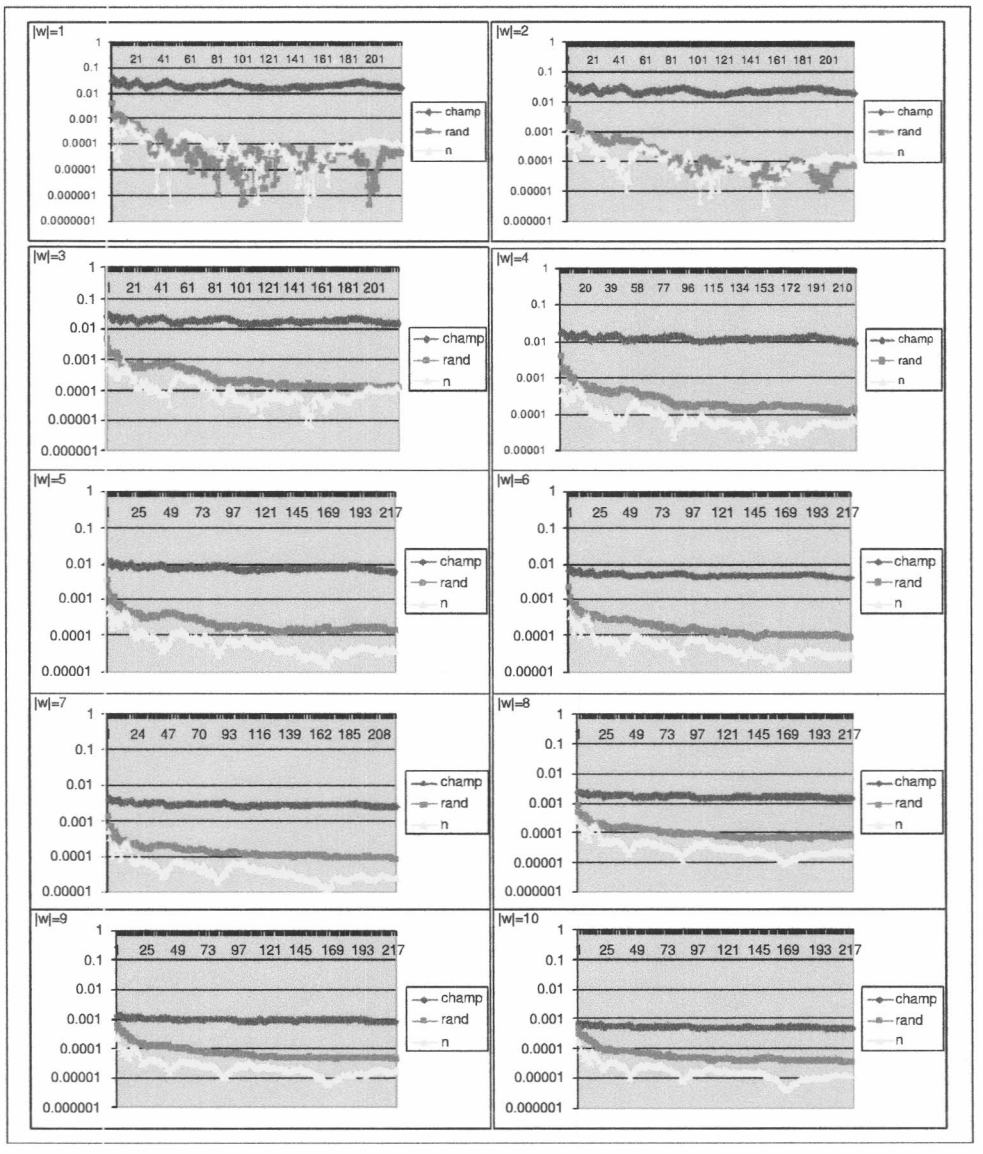


Figura 8: Distancias máximas a la frecuencia normal para bloques de largo 1 a 10.

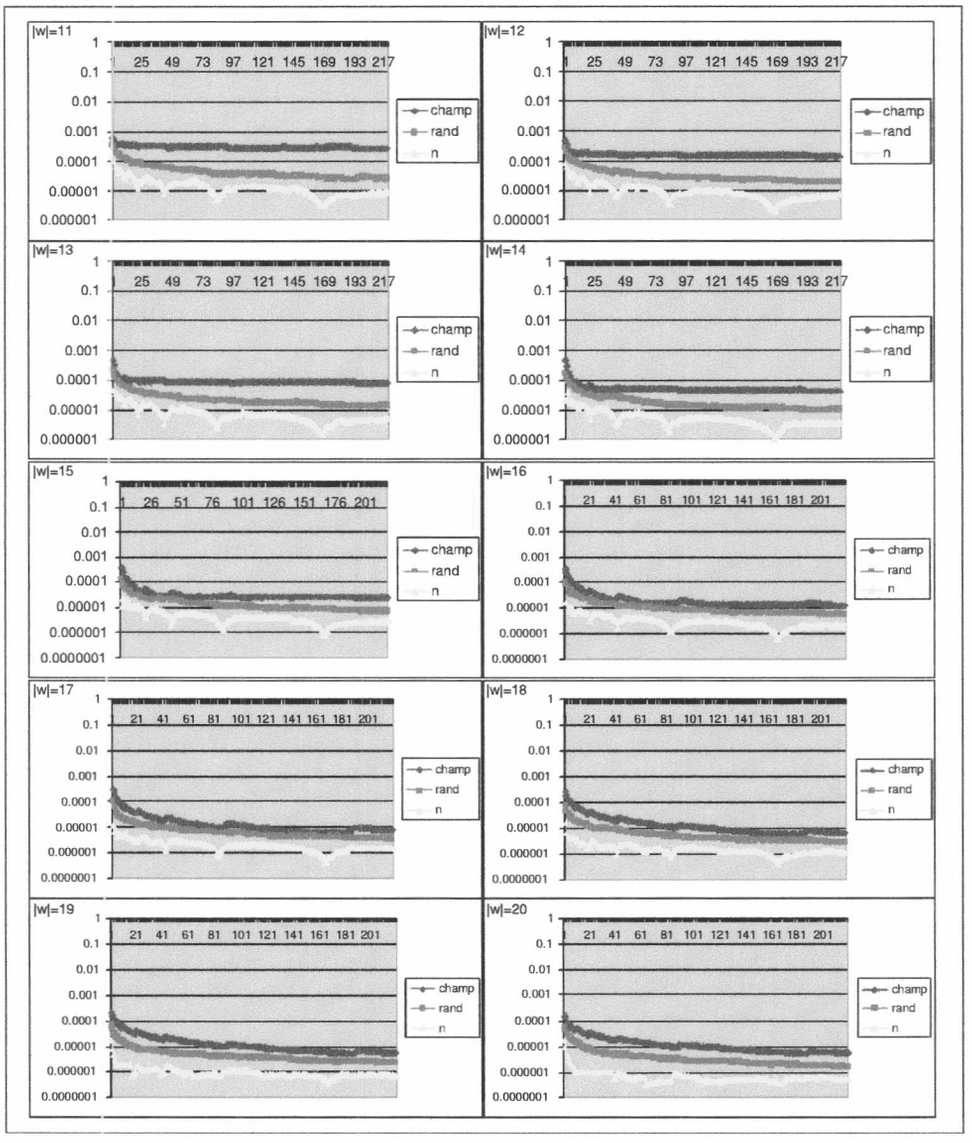


Figura 9: Distancias máximas a la frecuencia normal para bloques de largo 11 a 20.

Largo w	Champ	N	Rand
1	3	2	3
2	7	6	7
3	19	11	18
4	51	26	31
5	131	50	238
6	323	81	249
7	771	166	541
8	1795	475	2043
9	4099	693	3448
10	9219	1248	6883
11	20483	3637	16507
12	45059	6433	34043
13	98307	11298	76194
14	212995	32379	178997
15	458755	42905	342025
16	983043	117324	744254
17	2097155	225379	1833991
18	4456451	514615	3116436
19	9437187	842882	6893461

Cuadro 10: Para cada largo de palabra entre 1 y 19, tamaño del prefijo mínimo de n, *champ* y *rand* que incluye a todas las palabras de tal largo.

7.4.1. Aparición de nuevas palabras

Nuestro algoritmo genera SMS símbolo a símbolo buscando hacer aparecer la mayor cantidad de palabras distintas lo antes posible. Es interesante comparar el largo del prefijo mínimo que incluye a cada palabra de largo k para n, *champ* y *rand*. En el Cuadro 10 se puede comparar la velocidad de aparición de todas las palabras de largo 1 a 19 en las tres muestras. Se ve la misma relación que en los gráficos de límite de normalidad: n es la que necesita menos espacio para hacer aparecer todas las palabras de largo k, seguida por *rand*, y por último *champ*.

En la Figura 11 se puede comparar fácilmente la proporción de prefijo necesario para la aparición de todas las palabras de largo 1 a 19. Por ejemplo, el largo del prefijo de la SMS que contiene todas las palabras de 19 dígitos es un 8 % del largo del prefijo equivalente de la secuencia de Champernowne. El gráfico sugiere que las proporciones disminuyen a medida que se aumenta el largo de palabra.

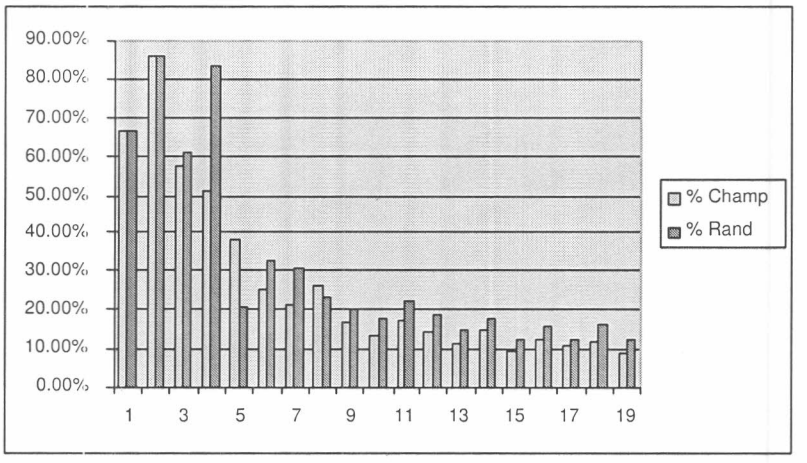


Figura 11: Proporción del tamaño de los prefijos de n que incluyen a todas las palabras de largo entre 1 y 19, con respecto a los mismos de las muestras de *champ* y *rand*.

7.5. Patrones en SMSs de bases mayores a 2

En la Figura 3 hemos graficado los primeros 1024^2 dígitos de n, y no observamos a simple vista irregularidades o patrones en su disposición. Es interesante repetir el experimento sobre SMSs generadas en bases mayores a 2.

Los gráficos de las Figuras 12 y 13 fueron generados a partir de los primeros dígitos de las SMSs de base 3 y 10 respectivamente, con función de decisión $f(i, Posibles) = min(Posibles)$. En estos gráficos sí se observan claramente zonas o franjas donde ciertos colores (dígitos) predominan sobre otros. Esto nos sugiere que en bases superiores a 2, y con el criterio de desición de elegir el mínimo dígito posible, las regularidades de las SMS son mucho más obvias que en el caso binario.

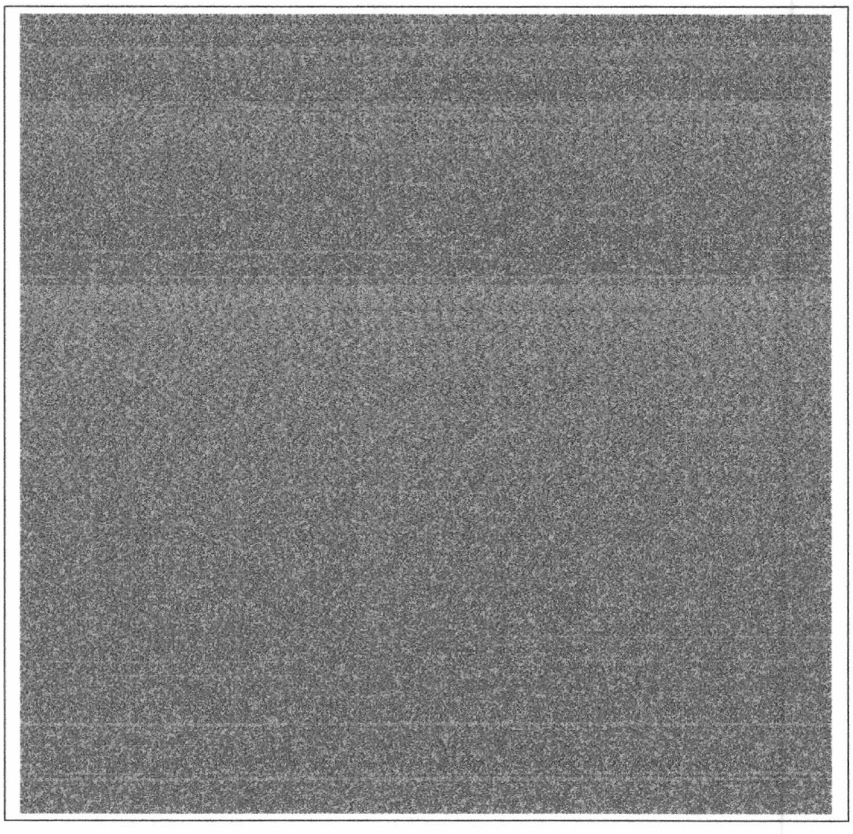


Figura 12: Los 729^2 primeros dígitos de la SMS en base 3 con función de decisión $f(i, Posibles) = min(Posibles)$

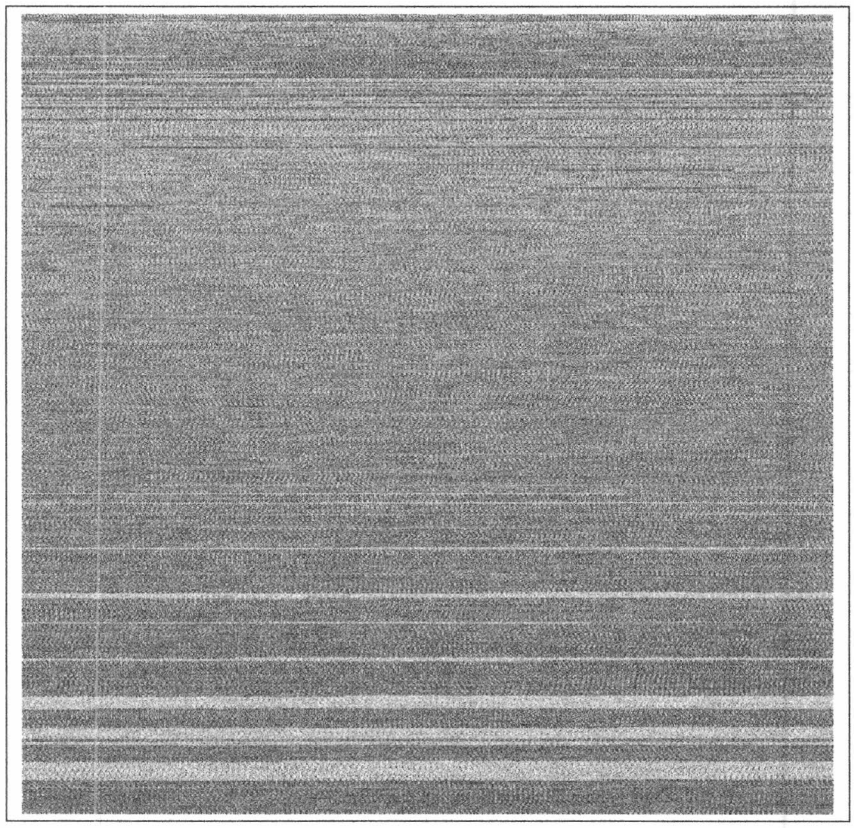


Figura 13: Los 1000^2 primeros dígitos de la SMS en base 10 con función de decisión $f(i, Posibles) = min(Posibles)$

8. Posdata: Antecedentes y conclusiones

La definición de las SMS nos parecía demasiado simple como para que no se le hubiese ocurrido a alguien más antes que a nosotros. Sin embargo, durante la mayor parte del tiempo del desarrollo de este trabajo, las búsquedas de publicaciones o referencias a secuencias de esta familia fueron infructuosas.

Ya finalizadas las demostraciones y experimentos, y en la etapa de redacción de este documento, encontramos dos casos independientes de definición de SMS binarias, con ejemplos específicos. Vale la pena hacer un breve comentario de cada uno.

Secuencia de Ehrenfeucht-Mycielsky

En American Mathematical Monthly [8], Ehrenfeucht y Mycielsky se plantean el problema de predecir el dígito s_{n+1} a partir de una secuencia binaria $s_1 s_2 \dots s_n$. Y proponen lo que llaman "metodo M" para arriesgar una "predicción". Informalmente, el método M sugiere s_{n+1} a partir de buscar el mayor sufijo $s_j s_{j+1} \dots s_n$ tal que ocurrió anteriormente en $s_1 s_2 \dots s_n$, y suponer que s_{n+1} también seguirá el mismo patrón. Es decir, para el máximo $n-j$ tal que $(s_j s_{j+1} \dots s_n) = (s_{j-i} s_{j+1-i} \dots s_{n-i})$, con mínimo $i > 0$ sugerir que s_{n+1} será igual a s_{n-i+1}. Los autores señalan que este método se parece a la forma en que los organismos vivos aprenden basándose en su experiencia.

A partir de esta definición, proponen una secuencia de dígitos binarios, que en el sentido del método M es "la más impredecible". Esta secuencia comienza con $s_0 = 0$ y se va construyendo contradiciendo la predicción sugerida por el método M. De esta manera, los primeros dígitos de la secuencia de Ehrenfeucht-Mycielsky son:

$$0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, \dots$$

Es fácil ver que esta secuencia es una SMS, ya que es un caso particular de lo que llamamos una Secuencia Completa binaria (definición 4). Es una secuencia binaria donde las primeras dos apariciones de toda subcadena son sucedidas por símbolos distintos.

En el mismo artículo hay una demostración de que la secuencia de Ehrenfeucht-Mycielsky es disyuntiva, y se deja abierta la pregunta de cuán aleatoria es esta secuencia desde un punto de vista estadístico [2].

En su nota al final del artículo el editor I. J. Good conjetura que seguramente la secuencia de Ehrenfeucht-Mycielsky es "flatter than flat-random", es decir, más uniforme en la distribución de subcadenas que una secuencia

[2] Sin conocer este artículo, nosotros nos habíamos preguntado lo mismo al ver los dígitos de nuestros ejemplos de SMS binarias, por lo que elegimos ponerle a esta tesis el mismo subtítulo que el del artículo de Ehrenfeucht y Mycielsky.

realmente aleatoria. Además da una definición informal de las SMS binarias, similar a nuestra definición de SMLNS, y propone llamarlas "Gambler's Fallacy sequences" ("secuencias de la falacia del jugador"). Dicha falacia consiste en suponer que, por ejemplo en la ruleta, luego de una cadena constante de "rojos" la probabilidad de "negro" es mayor, cuando en realidad siempre es la misma para ambos colores.

Secuencia de Kimberling

En 1997, sin conocer el artículo de Ehrenfeucht y Mycielsky, Clark Kimberling propuso en la revista Crux Mathematicorum un problema donde se definen las SMS binarias, según el criterio de minimizar la cantidad de repeticiones . El problema formulado consistía en decidir si estas "repetition-resistant sequences" son disyuntivas. Coincidentemente, como ejemplo particular de su secuencia da la misma SMS que utilizamos en nuestros tests de la sección 7. Nuestra demostración de disyuntividad fue la primera respuesta al problema, y fue publicada en la edición de Septiembre de 2003 [6]. Adjuntamos en el apéndice de esta tesis una copia de la publicación.

Nuestro aporte

A pesar de que existen algunos antecedentes de definición de SMS, nuestra tesis hace algunos aportes que generalizan el concepto. Aunque tanto la definición de Kimberling como la de Ehrenfeucht-Mycielsky están documentadas, hasta este momento no se había notado la relación entre ellas como para considerarlas de la misma familia.

En este trabajo damos tres criterios para definir las SMS, y demostramos su equivalencia. Estas definiciones no están limitadas a secuencias binarias sino que sirven para generar cualquier SMS con símbolos de un alfabeto finito. Adicionalmente, damos una demostración de disyuntividad que tampoco depende de la base utilizada. Mostramos un algorítmo que permite obtener varios millones de términos en pocos minutos utilizando una PC promedio. Y efectuamos diversos tests estadísticos que muestran las semejanzas y diferencias de los primeros millones de digitos de una SMS binaria con los primeros millones de dígitos de una secuencia normal y de una secuencia estadísticamente aleatoria.

Referencias

[1] D. H. Bailey and R. E. Crandall. On the random character of fundamental constant expansions. *Experimental Mathematics*, 10:2:175–190, 2001.

[2] E. Borel. Les probabilités dénombrables et leurs applications arthmetiques. *Rend. Circ. Mat. Palermo*, 27:247–271, 1909.

[3] G. Chaitin. *Algoritmic Information Theory.* Cambridge University Press, 1987.

[4] D. G. Champernowne. The construction of decimals normal in the scale of ten. *Journal of the London Mathematical Society*, 8:254–260, 1933.

[5] D. Copeland and P. Erdos. Note on normal numbers. *Bull. Amer. Math. Soc.*, 52:857–860, 1946.

[6] A. Dau. Solution to problem 2289. *Crux Mathematicorum with Mathematical Mayhem*, 29:320, 2003.

[7] N. G. de Bruijn. A combinatorial problem. *Proc. Kon. Ned. Akad. v. Wetensch.*, 49:758–764, 1946.

[8] A. Ehrenfeucht and J. Mycielski. A pseudorandom sequence. how random is it? *American Math. Monthly*, 99:371–375, 1992.

[9] C. Kimberling. About repetition-resistant sequence - problem 2289. *Crux Mathematicorum with Mathematical Mayhem*, 23:501, 1997.

[10] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, Reading, Massachusetts, third edition, 1998.

[11] G. Marsaglia. *The Marsaglia random number CDROM including the diehard battery of tests of randomness.* http://stat.fsu.edu/pub/diehard/, 1993.

[12] M. H. Martin. A problem of arrangements. *Bull. Amer. Math. Soc.*, 40:859–864, 1934.

9. Apéndices

9.1. Algunas SMSs en la Encyclopedia of Integer Sequences

9.2. Copia de nuestra demostración de disyuntividad publicada en Crux Mathematicorum

AT&T Integer Sequences RESEARCH

Greetings from the On-Line Encyclopedia of Integer Sequences!

Here is Sequence A093691 (this will take a moment):

ID Number: A093691
URL: http://www.research.att.com/projects/OEIS?Anum=A093691
Sequence: 0,1,2,3,4,5,6,7,8,9,0,0,2,0,3,0,4,0,5,0,6,0,7,0,8,0,9,1,0,0,
0,1,1,3,1,4,1,5,1,6,1,7,1,8,1,9,2,1,0,1,0,2,2,4,2,5,2,6,2,7,
2,8,2,9,3,2,0,0,3,3,5,3,6,3,7,3,8,3,9,4,3,0,0,4,4,6,4,7,4,8,
4,9,5,4,0,0,5,5,7,5,8,5,9,6,5
Name: a(n) is chosen so as to maximize the number of different substrings in (a(1),a(2),...,a(n)).
Comments: A "substrings maximizer sequence" (or SMS) in base 10.
If there is more than one choice for a(n), take the smallest.
References A. Dau, Secuencias Maximizadoras de Subcadenas, Tesis de licenciatura Cs de la Computacion, FCEyN, Universidad de Buenos Air (2004).
Links: A. Dau Secuencia Maximizador a de Subcadenas (Interactive Java generat
See also: Examples of SMS's in base 2: A079101, A038219, A093692.
Cf. A093692, A093693, A079101, A038219, A080415.
Keywords: nonn,new
Offset: 1
Author(s): Alejandro V. Dau (ad1n(AT)dc.uba.ar), Apr 11 2004

Show internal format for above sequence? Yes

Lookup | Welcome | Francais | Demos | Index | Browse | More | WebCam
Contribute new seq. or comment | Format | Transforms | Puzzles | Hot | Classics
More pages | Superseeker | Maintained by N. J. A. Sloane (njas@research.att.com)

home | people | projects | research areas | resources |

Figura 14: SMS en base 10, con función de decisión *min*

AT&T Integer Sequences RESEARCH

Greetings from the On-Line Encyclopedia of Integer Sequences!

Here is Sequence A093692 (this will take a moment):

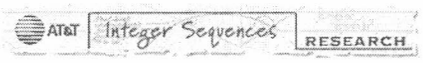
ID Number: A093692
URL: http://www.research.att.com/projects/OEIS?Anum=A093692
Sequence: 0,1,1,0,0,0,1,0,1,1,1,0,1,0,0,1,1,1,1,0,0,1,0,0,0,0,1,1,0,1,
1,0,0,1,1,0,0,1,0,1,0,1,0,0,0,1,1,1,0,0,0,0,0,1,0,0,1,0,0,1,
1,0,1,0,1,1,0,1,1,1,1,1,1,0,1,1,1,0,0,1,1,1,0,1,1,0,1,0,0,0,0,0,
0,0,0,1,1,1,1,1,1,1,1,0,0,0,1,1
Name: a(n) is chosen so as to maximize the number of different substrings in (a(1),a(2),...,a(n)) (in base 2).
Comments: A "substrings maximizer sequence" (or SMS) in base 2.
See also: See A093691 for more information.
Cf. A093691, A093693, A079101, A038219, A080415.
Keywords: nonn,new
Offset: 1
Author(s): Alejandro V. Dau (adln(AT)dc.uba.ar), Apr 11 2004

Show internal format for above sequence? Yes

Lookup | Welcome | Francais | Demos | Index | Browse | More | WebCam
Contribute new seq. or comment | Format | Transforms | Puzzles | Hot | Classics
More pages | Superseeker | Maintained by N. J. A. Sloane (njas@research.att.com)

home | people | projects | research areas | resources |

Figura 15: SMS en base 2, con función de decisión devolviendo alternativamente 0 y 1



Greetings from the On-Line Encyclopedia of Integer Sequences!

Here is Sequence A093693 (this will take a moment):

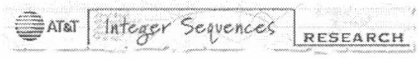
ID Number: A093693
URL: http://www.research.att.com/projects/OEIS?Anum=A093693
Sequence: 0,1,2,0,0,2,1,0,0,0,1,1,0,1,0,2,2,0,1,0,0,1,0,1,1,1,2,1,1,0,
0,2,0,2,0,0,0,0,2,2,1,2,2,2,0,0,1,2,1,0,1,2,2,0,2,1,1,1,0,2,
0,1,1,2,0,1,2,0,2,2,2,1,0,2,1,2,0,0,0,1,0,0,0,0,0,1,2,0,1,0,
1,0,0,2,1,1,2,2,1,1,0,1,1,0,0
Name: a(n) is chosen so as to maximize the number of different substrings in (a(1),a(2),...,a(n)) (in base 3).
Comments: A "substrings maximizer sequence" (or SMS) in base 3.
See also: See A093691 for more information.
Cf. A093691, A093692, A079101, A038219, A080415.
Keywords: nonn,new
Offset: 1
Author(s): Alejandro V. Dau (ad1n(AT)dc.uba.ar), Apr 11 2004

Show internal format for above sequence? Yes

Lookup | Welcome | Francais | Demos | Index | Browse | More | WebCam
Contribute new seq. or comment | Format | Transforms | Puzzles | Hot | Classics
More pages | Superseeker | Maintained by N. J. A. Sloane (njas@research.att.com)

home | people | projects | research areas | resources |

Figura 16: SMS en base 3, con función de decisión *min*



Greetings from the On-Line Encyclopedia of Integer Sequences!

Here is Sequence A079101 (this will take a moment):

ID Number: A079101
URL: http://www.research.att.com/projects/OEIS?Anum=A079101
Sequence: 0,1,0,0,0,1,1,0,1,0,1,1,1,0,0,1,0,0,1,1,1,1,1,0,1,1,0,0,0,0,0,
1,0,1,0,0,0,0,1,1,1,0,1,0,0,1,0,1,1,0,1,1,1,1,1,1,0,0,0,1,0,0,
0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,1,1,0,1,0,1,0,0,1,1,0,0,1,1,0,1,0,
1,0,1,1,0,0,1,1,0,1,1,0,1,0,0
Name: A repetition-resistant sequence.
Comments: a(n) = 0 or 1, chosen so as to maximize the number of different subsequences that are formed.
a(n+1)=1 if and only if (a(1),a(2),...,a(n),0), but not (a(1),a(2),...,a(n),1), has greater length of longest repeated segment than (a(1),a(2),...,a(n)) has.
In February, 2003, Alejandro Dau solved Problem 3 on the Unsolved Problems and Rewards website, thus establishing that every binary word occurs infinitely many times in this sequence.
References C. Kimberling, Problem 2289, Crux Mathematicorum 23 (1997) 501.
Links: A. Dau Secuencia Maximizadora de Subcadenas (Interactive Javagenerator
C. Kimberling, Unsolved Problems and Rewards.
Example: a(7)=1 because (0,1,0,0,0,1,0) has repeated segment (0,1,0) of length 3, whereas (0,1,0,0,0,1,1) has no repeated segment of length
See also: Cf. A079136, A079335, A079336, A079337, A079338.
Sequence in context: A030213 A071037 A071038 this_sequence A076478 A091444 A091447
Adjacent sequences: A079098 A079099 A079100 this_sequence A079102 A079103 A079104
Keywords: nonn
Offset: 1
Author(s): Clark Kimberling (ck6(AT)evansville.edu), Jan 03 2003
Show internal format for above sequence? Yes

Lookup | Welcome | Francais | Demos | Index | Browse | More | WebCam
Contribute new seq. or comment | Format | Transforms | Puzzles | Hot | Classics
More pages | Superseeker | Maintained by N. J. A. Sloane (njas@research.att.com)

home | people | projects | research areas | resources |

Figura 17: Secuencia de Kimberling, SMS en base 2 con función de decisión constante 0

Crux Mathematicorum 29 · #5 September 2003

SOLUTIONS

No problem is ever permanently closed. The editor is always pleased to consider for publication new solutions or new insights on past problems.

We apologise for omitting the name of ANDREI SIMION, student, Cooper Union for Advancement of Science and Art, New York, NY, USA from the list of solvers of 2717 and 2718, and the name of MICHEL BATAILLE, Rouen, France from the list of solvers of 2729.

2289*. [1997 : 501] *Proposed by Clark Kimberling, Evansville, IN, USA.*

Use any sequence, $\{c_k\}$, of 0's and 1's to define a *repetition-resistant sequence* $s = \{s_k\}$ inductively as follows:

1. $s_1 = c_1$, $s_2 = 1 - s_1$;
2. for $n \geq 2$, let

$$\begin{aligned} L &= \max\{i \geq 1 : (s_{m-i+2}, \ldots, s_m, s_{m+1}) \\ &\qquad = (s_{n-i+2}, \ldots, s_n, 0) \quad \text{for some } m < n\}, \\ L' &= \max\{i \geq 1 : (s_{m-i+2}, \ldots, s_m, s_{m+1}) \\ &\qquad = (s_{n-i+2}, \ldots, s_n, 1) \quad \text{for some } m < n\}. \end{aligned}$$

(so that L is the maximal length of the tail-sequence of $(s_1, s_2, \ldots, s_n, 0)$ that already occurs in $(s_1, s_2, \ldots, s_n)$, and similarly for L'), and

$$s_{n+1} = \begin{cases} 0 & \text{if } L < L', \\ 1 & \text{if } L > L', \\ c_n & \text{if } L = L'. \end{cases}$$

(For example, if $c_i = 0$ for all i, then

$$s = (0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0,$$
$$1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, \ldots)\ )$$

Prove or disprove that s contains every binary word.

Solution by Alejandro Dau, Universidad de Buenos Aires, Buenos Aires, Argentina, modified slightly by the editors.

We will prove that every binary word occurs infinitely many times in s. For any symbol $\sigma \in \{0, 1\}$, we write $\bar{\sigma}$ for the opposite symbol.

We first show that the words of length 1 appear infinitely many times; that is, the symbols 0 and 1 each appear infinitely often in s. Each symbol appears at least once, because, by definition, s starts with 01 or 10. Suppose the symbol σ appears just finitely many times, the last time being at s_k.

Then $s_{k+i} = \overline{\sigma}$ for all $i \geq 1$. In particular, $s_{2k+2} = \overline{\sigma}$. But this contradicts the definition of s, because the string $s_{k+2} \cdots s_{2k+2} = \overline{\sigma}^{k+1}$ already occurs in $s_1 \cdots s_{2k+1}$ as $s_{k+1} \cdots s_{2k+1}$, whereas the word $\overline{\sigma}^k\sigma$ does not occur as a substring of $s_1 \cdots s_{2k+1}$. We conclude that each word of length 1 appears infinitely often in s.

Now suppose there is some word of length greater than 1 that appears only finitely often. Then there is a shortest length $n \geq 2$ for such words. Choose some word of length n that appears only finitely often, say $t = d_1 \ldots d_n$, and let $k \geq 0$ be the number of times that t appears. The word $d_1 \ldots d_{n-1}$ appears infinitely often, because its length is less than n. Letting $t^* = d_1 \ldots d_{n-1}\overline{d_n}$, we see that t^* must appear infinitely often, since t appears only finitely often.

Suppose $k = 0$. Then the word t does not appear at all in s. Let the first two appearances of t^* end at positions i and j, respectively. Thus, $s_{i-n+1} \cdots s_i = t^*$ (the first appearance) and $s_{j-n+1} \cdots s_j = t^*$ (the second appearance). The fact that $s_j = \overline{d_n}$ contradicts the definition of s, because the word t^* already occurs in $s_1 \cdots s_{j-1}$ as $s_{i-n+1} \cdots s_i$, whereas the word t does not occur in $s_1 \cdots s_{j-1}$.

Suppose $k \geq 1$. Then the word t appears at least once in s. Let the last appearance of t end at position p. Thus, $s_{p-n+1} \cdots s_p = t$, and this is the last appearance of t in s. Consequently, there are no words in s of length $p+1$ that end with the string t. On the other hand, there must be some word of length $p+1$ ending in t^* that occurs more than once in s, because there are infinitely many appearances of t^* in s and only finitely many words of length $p+1$. Thus, there must exist two strings $s_{i-p} \cdots s_i$ and $s_{j-p} \cdots s_j$, with $i < j$, that represent the same word w ending in t^*. We have $s_j = \overline{d_n}$. But the definition of s requires that $s_j = d_n$, because the tail-sequence w ending in t^* already occurs in $s_1 \cdots s_{j-1}$ as $s_{i-p} \cdots s_i$, whereas there are no words in s of length $p+1$ ending in t. We have a contradiction.

A contradiction comes from assuming that a word appears in s only a finite number of times. Therefore, every word appears infinitely many times.

2664. [2001 : 403] *Remark : In his solution of a generalization of this problem, Walther Janous proved in* [2002 : 410] *the following result :*

Let a, b, c be non-negative reals and let $r \in \mathbb{R}$ with $r \geq 2$.

Then

$$a^r(b+c) + b^r(c+a) + c^r(a+b) \geq \frac{2}{3^{\frac{r-2}{2}}}(ab+bc+ca)^{\frac{r+1}{2}}. \quad (1)$$

At the end of his solution he made the remark :

Clearly equality holds in our theorem if $r = 1$. This leads to the natural question : What happens for $r \in (1, 2)$?

9.3. Listados

9.3.1. Generador de SMS - subsubMultibase.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>

// constantes para acotar el uso de memoria (y el largo de la secuencia)
#define MAXMEM 100000000
#define MAXGEN 100000000
#define MAXB 255

typedef unsigned char simbolo;

typedef struct marcas {
        char *a;                /* las marcas */
        unsigned ml;            /* el mximo largo marcado */
        unsigned b;             /* la base */
} marcas;

typedef struct palabra {
        simbolo *p;
        unsigned lp;
} palabra;


unsigned long intpow(unsigned b, unsigned e) {
        unsigned long a=1;
        while (e > 0) {
                a *= b;
                e--;
        }
        return a;
}



marcas *init(unsigned b) {
        unsigned long mm,ma;
        char *a;
        struct marcas *m;
        unsigned ml,i;
```

```
        assert (b > 1);
        assert (MAXMEM > b);
        i=ml=0;
        mm=ma=b;
        while (ma < MAXMEM) {
                mm = ma;
                ml = i;
                i++;
                ma += b * intpow (b, i);
        }

        a = malloc(mm * sizeof(char));
        assert (a!=NULL);

        fprintf(stderr, " inicializando %lu marcas de cadenas de l<=%u..", mm, ml);
        for (i=0; i < mm; i++)
                a[i]=0;

        fprintf(stderr, ".\n");

        m=malloc ( sizeof(struct marcas));
        assert (m!=NULL);

        assert (ml > 0);
        m->ml = ml;
        m->b = b;
        m->a = a;
        return m;
}

simbolo decidir (unsigned n, palabra *p ) {
        /* Funcin de decisin
         * . p tiene una lista de smbolos sobre la que se debe decidir
         * . n es la posicin en la SMS, que podra ser usada para
         *   dar ms libertad en la eleccin del dgito
         *   no es usada en este caso, simplemente se elige el mnimo smbolo */

        unsigned min=MAXB; /* base mxima */
        unsigned i=0;

        while (i < p->lp) {
                if ( (p->p)[i] < min )
                        min = (p->p)[i];
```

```
                i++;
        }
        return min;
}


unsigned long val(unsigned b, palabra *p) {
        unsigned long a,lp;
        simbolo *pp;

        a=0;
        lp=p->lp;
        pp=p->p;
        while ( lp != 0 ) {
                a *= b;
                a += *pp;
                pp++;
                lp--;
        }
        return a;

}

unsigned long pos(unsigned b, palabra *p) {
        return (b * (intpow(b, p->lp)-1)/ (b-1) + val(b, p) * b);
}

void marcar( marcas *m, palabra *p, simbolo d ) {
        assert (m->b > d);
        if (m->ml < p->lp) {
                printf ("demasiados dgitos para la actual base de marcas\n");
                exit(-1);
        }
        m->a[pos(m->b, p) + d] = 1;
}

char esmarcado( marcas *m, palabra *p, simbolo d ) {
        assert (m->b > d);
        if (m->ml < p->lp) {
                printf ("demasiados dgitos para la actual base de marcas?\n");
                exit(-1);
        }
        return m->a[pos(m->b, p) + d];
}
```

```
simbolo decidirStruc ( unsigned ls , marcas *m, palabra *p ) {
        palabra *a;
        unsigned i , j ;
        simbolo ap[MAXB];
        simbolo d;
        palabra aa;


        j=0;
        a = &aa;
        a->p = ap;
        for ( i=0; i < m->b; i++) {
                if  (! esmarcado(m,p,i)) {
                        a->p[j++]=i;
                }
        }
        a->lp = j;

        d=decidir( ls , a );
        return d;
}


/*********************************************/

void usage(char *p) {
        printf (" %s <base>\n", p);
        printf  (" donde 1 < base < 256\n");
        return;
}


// Main

int  main (int argc , char *argv []) {

marcas *a;
int  b;
unsigned l, i , ls ;
simbolo *s;
simbolo d;
palabra pp;
palabra *p;
```

```
    s = malloc( MAXGEN * sizeof(simbolo));
    assert (s != NULL);
    ls = 0;


    if (argc < 2 || ( atoi(argv [1]) < 2 || atoi(argv[1]) > MAXB-1) ) {
        usage(argv [0]);
        return -1;
    }
    b=atoi(argv[1]);
    a=init(b);

    i=0;

    pp.p=s;
    pp.lp=0;

    p=&pp;

    while (1) {
        l=0;
        while (1) {
            assert( ls >= l );
            assert ( l <= a->ml );
            p->lp=l; p->p=s+(ls-l);
            d = decidirStruc ( ls , a, p );
//          printf( "decidido: %u ", d );
            if (d < MAXB) break;
            l++;
        };

        for (l=0 ; l <= a->ml; l++) {
            if (l > ls) break;
            p->lp=l; p->p=s+(ls-l);
            marcar(a, p, d);
        }
        s[ls++]=d;
        assert(ls < MAXGEN);
        printf(" %u", d);
    }

    return 0;
}
```

9.3.2. permut.pl - Programa perl usado para testear aleatoriedad mediante el método de permutaciones (Knuth [10])

```
#! /usr/bin/perl -w

# Calcula chisq de las combinaciones en el orden de a tres palabras.
# p1 < p3 < p2
# p2 = p3 < p1
# ... etc

use ChiSq;

sub acumular_perm {
        my $cs = shift;
        my $l = shift;
        my @s = @_;

        @orden = sort @s;

        my $i = 0;
        my $res = "";

        $l = scalar @s;

        while ( $i < $l ) {
                my $j = 0;
                while ( $j < $l ) {
                        if ($orden[$i] eq $s[$j]) {
                                $res .= $j+1;
                                last;
                        }
                        $j++;
                }
                $i++;
        }
        $cs->account($res);

        return;


}

my $l = 3;
my $b = shift @ARGV;
```

```
if (! defined $b) {
        print "forma de uso: $0 <largo de palabra> \n";
        exit -1;
}
my $cs = new ChiSq;
$/ = \1;

# probabilidad de las tres palabras iguales:
# asumo que las palabras son base dos
my $bp = 2 ** $b;
my $p1 = 1/($bp ** 2);
# probabilidad de dos iguales y una distinta:
my $p2 = ( ($bp-1) * 2) /($bp ** 2) + ($bp-1) / ($bp ** 2) ;
# probabilidad de tres distintas:
my $p6 = (($bp-1) * ($bp-2))/($bp ** 2)/6;


my @a = ();
my $s = "";
while (<>) {
        s/[^0-9A-Za-z]//g;
        next if ($_ eq "");
        $s .= $_;
        if (length($s) == $b) {
                push @a,$s;
                $s="";
                if (scalar @a == 3) {
                        acumular_perm($cs, $l, @a);
                        @a = ();
                        $cant++;
                }
        }
}

my $tabla = {
        "123" => $p6,
        "213" => $p6,
        "132" => $p6,
        "321" => $p6,
        "231" => $p6,
        "312" => $p6,
        "211" => $p2/6,
        "311" => $p2/6,
        "112" => $p2/6,
```

```
        "113" => $p2/6,
        "122" => $p2/6,
        "221" => $p2/6,
        "111" => $p1 };

my $df = scalar keys( %{$tabla} ) - 1;

my $val = $cs->eval_distrib($tabla) ;
print "cant:\t$cant\tchi:\t$val\t";
print "p:\t" . ChiSq::calc_x_df($val, $df) . "\n";
```