UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

Colegio de Ciencia y Tecnología

Modelo de un token de ocho dígitos y pruebas de aleatoriedad.

T E S I S

PARA OBTENER EL TÍTULO DE:

Licenciada en Modelación Matemática

PRESENTA:

Ana Karen Ramírez López

DIRECTOR

M en C. Juan Carlos Aguilar Franco

Ciudad de México, mayo de 2021

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

CA Coordinación Académica

SISTEMA BIBLIOTECARIO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO
COORDINACIÓN ACADÉMICA

RESTRICCIONES DE USO PARA LAS TESIS DIGITALES

DERECHOS RESERVADOS©

AGRADECIMIENTOS

"Si la gente no cree que las Matemáticas
son simples, es sólo porque no se dan cuenta
de lo complicada que es la vida."
John Von Neumann

Gracias a la Universidad Autónoma de la Ciudad de México y al Colegio de Ciencia y Tecnología por fórmarme en ella. Gracias a todas las personas que fueron partícipes de este proceso, a todos mis profesores que conocí durante mi formación, gracias por su tiempo, por su apoyo, así como su sabiduría que me transmitieron, gracias a todos aquellos que me ayudaron, ya sea de manera directa o indirecta, a todos aquellos que indirectamente me hicieron tropezar, gracias por que aprendí que nada es fácil y siempre hay que esforzarse cada día, gracias a todos por su pequeño aporte que hoy se ve reflejado en la culminación de mi paso por la Universidad.

DEDICATORIA

Dedico este trabajo principalmente a Dios por haberme dejado ver cada día un nuevo amanecer, darme la fuerza y el valor de haber llegado hasta este momento tan importante de mi formación profesional.

A mi mamá por ser una gran mujer, por ser uno de los pilares de mi hogar y por siempre estar conmigo, apoyarme en cada momento, en cada situación y sobretodo en cada decisión que tome, por demostrarme su apoyo y cariño incondicional sin importar nuestras diferencias y opiniones. A mi papá por ser el pilar más fuerte de mi hogar, por que a pesar de estar trabajando, siempre cuento con su apoyo y cariño. A mi hermano por su apoyo que siempre fue mutuo, por aguantarme en cada momento que pase, por sus opiniones, por alentarme a seguir. Gracias a toda mi familia por estar siempre conmigo.

A mis tías, tíos, primos y primas, a mi familia en general, por haberme brindado su apoyo, por preguntar como me iba, por alentarme a seguir con mi carrera, por motivarme a seguir adelante, por compartir conmigo buenos y malos momentos.

También doy las gracias a tres personas muy especiales en mi vida, que ya no se encuentran conmigo, a mis abuelitos paternos y a mi abuelita materna, con cada uno tengo muchos recuerdos y ellos siempre me alentaron a seguir estudiando, a seguir preparándome, confiaban demasiado en mi, gracias por todo, los llevare siempre en mi corazón los AMO.

A mis amigos y compañeros que tuve durante mi paso por la Universidad, gracias por el apoyo, los conocimientos que hicieron fortalecernos unos a otros, por cada experiencia vivida que hizo una de las mejores etapas, sin duda fue un placer conocerlos.

A mis lectores que son parte fundamental de esto, muchas gracias por formar parte de mi formación, por sus conocimientos que me transmitieron, por su tiempo y dedicación.

A mi director de tesis y tutor Juan Carlos Aguilar, gracias por apoyarme en todo momento, por su tiempo, su paciencia, por sus conocimientos, por hacer que mi formación se fuera construyendo a través de cada semestre, por hacer de esta etapa una de las mejores, por hacerme que me esforzara cada día, por que gracias a usted siempre puse todo mi esfuerzo y dedicación. Gracias por todas las charlas, por los consejos, por que más que un profesor fue un gran amigo.

Por último pero no menos importante gracias a estas personas, por que han estado durante este proceso apoyándome incondicionalmente: Karen, Luisa, Alejandra y Margarita gracias por todo, por siempre estar conmigo, en verdad se los agradezco por todo el apoyo que me brindaron durante todo este tiempo.

Gracias a todas las personas que llegaron a mi vida, a los que se fueron, gracias a todos por que gracias a ellos hicieron una mejor versión de mi.

A todos ustedes

¡Gracias!

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

OBJETIVO

El objetivo principal de este proyecto es darle el lugar especial que merecen los números aleatorios, pero en esta investigación es la generación de números pseudoaleatorios, que son generados por una función determinista, con ciertos parámetros, aunque es un tema que se ve a lo largo de la carrera en Matemáticas, no se le da la importancia, es más ni se menciona lo extremadamente necesario que es para varias áreas de la ingeniería.

Lo que busco hacer es mostrar las ventajas y desventajas de los diferentes métodos de generar **números pseudoaleatorios**, ver que gracias a ellos es fácil llevar una manipulación de estos números, siempre y cuando se entienda como funcionan el algoritmo.

Todo esto funciona gracias a unos generadores ya propuestos, estos algoritmos se propusieron en la década de los cuarenta y son un aporte para la investigación, dichos generadores fueron modificados aquí para generar números de ocho dígitos.

Todos los generadores serán sometidos a diferentes pruebas y de ahí se elegirá el "mejor" para ser implementado para el acceso de una página web, es decir se dará acceso a dicha página siempre que se ingrese la clave correcta, la cual será generada en tiempo real y de forma aleatoria.

Esta clave será generada a través de un token móvil, tiene la misma función que la de un "token bancario", y se ocupa para cada transacción que se realiza, el token dará una clave de acceso, solo si la persona conoce el token móvil logrará la autentificación correcta y obtendrá el acceso.

INTRODUCCIÓN

En el presente trabajo se hablará sobre los generadores de números pseudoaleatorios, para algunas áreas de estudio: las finanzas, estadísticas, criptografía entre muchas más, son importantes ya que con ellos se pueden hacer simulaciones para predecir posibles resultados en condiciones variadas o saber el comportamiento de las variables aleatorias, para así adaptarlo a la realidad e implementarlo.

Los números pseudoaleatorios se obtienen a partir de un generador, que produce una sucesión de valores que se **pretende** sean una secuencia de variables aleatorias independientes y distribuidas uniformemente y que no se repitan.

En este trabajo se presentarán los tipos de generadores más comunes, además de algunos ejemplos, también se aplicarán algunas pruebas de aleatoriedad (estadísticas) para los números pseudoaleatorios generados, las cuales el conjunto de números debe de cumplir.

La idea principal para generar números pseudoaleatorios, es que estos sean fáciles de reproducir, además de que deberán de cumplir con ciertas condiciones que se describirán con todo detalle en el desarrollo de este trabajo.

CAPÍTULO 1

ANTECEDENTES DE LOS NÚMEROS ALEATORIOS

SECCIÓN 1.1

Números aleatorios

Figura 1.1: Von Neumann
[1]

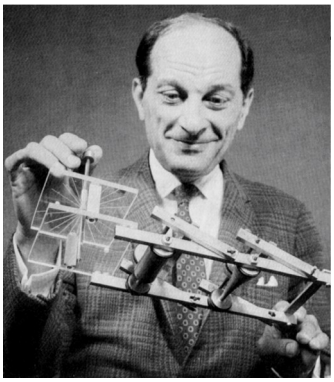


Figura 1.2: Stanislaw Ulam
[2]

[1]Neumann, V.(2008).John von Neumann.[Figura]. Recuperado de *https* : *//commons.wikimedia.org/wiki/File* : $JohnvonNeumann - LosAlamos.gif$

[2]Stanislaw, U.(2012).Stanislaw Ulam.[Figura]. Recuperado de *https* : *//en.wikiquote.org/wiki/File* : $STAN_ULAM_HOLDING_THE_FERMIAC.jpg$

En el siglo XVII, se desarrollaron distintos campos de estudio, uno de los principales e importantes fueron los juegos de azar, pues con ellos se representaron y simularon fenómenos aleatorios. En la década de los cuarenta se cita, "la construcción de modelos arranca desde el renacimiento, el uso moderno de la palabra simulación data de 1940, cuando los científicos Von Neuman y Ulam que trabajaban en el proyecto Montecarlo, durante la segunda guerra mundial, resolvieron problemas de reacciones nucleares cuya solución experimental sería muy cara y el análisis matemático demasiado complicado." [4, pág.11]

El método Montecarlo es un método numérico que permite resolver distintos tipos de problemas físicos y matemáticos mediante la simulación de variables aleatorias, y así proporcionar una solución aproximada a la realidad con mira a su implementación.

Como se mencionó anteriormente, el método de Montecarlo se usó para resolver los juegos de azar, un ejemplo clásico:

- Se lanza una moneda y las únicas opciones que existen son: sol y águila, la probabilidad que caiga sol es $\frac{1}{2}$ y si es águila de igual manera es $\frac{1}{2}$ pero, que pasaría si en vez de lanzar una moneda se lanzan 3, 5, o se lanzan n monedas. Entonces el método de montecarlo hace la simulación de lanzar $\boldsymbol{n}$ monedas y se observa los resultados que caen en el intervalo (0,1), el cual se divide en tantas partes iguales y así saber donde cayó "aleatoriamente".

El método Montecarlo fue bautizado así por su clara analogía con los juegos de la ruleta de los casinos, el más célebre de ellos es el de Montecarlo, casino cuya construcción fue propuesta en 1856 por el príncipe Carlos III de Mónaco, siendo inaugurado en 1861 [3].

El uso de los métodos de Montecarlo como herramienta de investigación, proviene del trabajo realizado en el desarrollo de la bomba atómica durante la segunda guerra mundial. Este trabajo con lleva la simulación de problemas probabilísticos en hidrodinámica, el requerimiento que se necesitaba en 1943, para el desarrollo de dicha bomba, eran los **números aleatorios**.

Así como se utilizaron los números aleatorio para la bomba nuclear, se utilizaron para otras cosas distintas, pero la pregunta frecuente que muchos se hacen es, ¿dónde más se ocupan estos números? O ¿Cómo son usados en esas áreas?, a continuación lo explico de manera breve.

1.1.1. ¿Dónde se utilizan los números aleatorios?

La necesidad de generar números aleatorios es porque se utilizan en distintos campos de estudio, por ejemplo:

- Simulaciones de muestreo: Aquí se utilizan para predecir, sacando alguna pequeña muestra y analizando su comportamiento.
- Finanzas: Aquí lo que corresponde es observar cómo se comportan los valores de las acciones, bolsa de valores, entre otros. Para después tomar decisiones, lo anterior se relaciona con (árboles de decisión o la teoría de juegos).
- Estadística: Aquí se utilizan tablas de números aleatorios, para ayudar a los científicos, ingenieros y estadísticos, que requieran trabajar con números aleatorios e independientes.
- Criptografía: En esta área lo que se necesita es tener una seguridad fuerte y confiable. Esto se vería reflejado en lo que se conoce como "**token**", donde por medio de un dispositivo o aplicación se generan números pseudoaleatorios, que sirven para proteger tus datos bancarios y cada vez que ingresas a ellos se generan nuevos números, esto principalmente en bancos.
- Programación: Los números pseudoaleatorios son necesarios para los algoritmos, en donde se necesite aleatoriedad.

Estos fueron algunos campos de estudio dónde se ocupan los números aleatorios, pero otra de las preguntas es, ¿son números aleatorios o números pseudoaleatorios?, puede llegar a ser un poco confuso, pero a continuación mencionaré cuál es la diferencia.

1.1.2. ¿Generar números aleatorios o pseudoaleatorios?

Para poder generar números aleatorios mediante la computadora, se utiliza una función en **java** que es *rand()* y siempre genera el mismo orden.

En cambio para generar números pseudoaleatorios, se realiza con ayuda de algún algoritmo con condiciones diferentes, se puede generar la cantidad necesaria de números pseudoaleatorios diferentes y así poder usar dichos números de manera confiable.

Una vez que se generaron los números pseudoaleatorios se deben de someter a pruebas de aleatoriedad para comprobar que el generador ocupado si es confiable, rápido y eficiente. Las pruebas que existen son:

- Prueba de medias (promedios)
- Prueba de varianzas
- Prueba para la uniformidad
- Bondad de ajuste o χ^2 (frecuencias)
- Prueba de Kolmogorov-Smirnov
- Corridas por arriba y por abajo del promedio
- Prueba de series
- Prueba del *poker*
- Entre otras

Más adelante se mencionarán cuáles son las pruebas que se van a emplear para este trabajo.

1.1.3. Ventajas y desventajas de los números aleatorios

Se presentan las ventajas y desventajas para obtener números aleatorios.

1. Manual: Lanzamiento de dados, monedas.
    - Ventaja: Las series obtenidas son aleatorias
    - Desventaja: son lentas de obtener y se requiere de memoria grande para almacenamiento.

2. Tablas: 1 hasta 100,000 números

   - En la RAND Corporation se publico en 1947, una tabla que ofrecen un millón de números aleatorios. Para análisis cientifico, juegos, sorteos, etc.
   - Ventaja: Son aleatorias y reproducibles.
   - Desventaja: Lentitud, requiere memoria grande para almacenamiento.

3. Computadora "Software"

   - Este método se usa para hacer que las simulaciones del generador sean más rápidas, pero depende de qué generador se utilice.
   - Ventaja: Son rápidas, requerimientos básicos, son reproducibles.
   - Desventaja: No son independientes, por que a la hora de ocupar un generador, lo que uno propone son parámetros y esos pueden ser manipulados y así obtener lo deseado. Por lo anterior ya no es **aleatorio** sino **pseudoaleatorio**

SECCIÓN 1.2

Definiciones

Las definiciones que se presentan a continuación son la base, para introducirse al tema de los números aleatorios y pseudoaleatorios, empezamos por las siguientes:

Definición

Generador

Es un algoritmo que produce una secuencia de números aleatorios o mejor conocidos como pseudoaleatorios, que simula una distribución uniforme e independiente, con ayuda de un número fijo determinado como valor inicial, llamado **semilla**.

Definición

Simulación

Es una técnica numérica, la cual ayuda a realizar experimentos en una computadora, con el fin de saber el comportamiento del mundo real y poder realizar predicciones en diferentes intervalos de tiempo.

Definición

Periodo

El periodo puede representarse como un ciclo, una vez generado el máximo de números simulados su ciclo a finalizado, si se genera un número mayor al periodo (parámetro) la semilla tenderá a repetirse.

Definición

Hipótesis

- Hipótesis nula

  Son proposiciones que niegan la relación entre variables, se denota con H_0

- Hipótesis alternativa

  Son posibilidades diferentes o alternas para ver la relación entre las variables, se denota con H_1

Definición

Números aleatorios

Estos son obtenidos al azar, tienen la misma probabilidad de ser elegidos y son independientes entre ellos.

Definición

Números Pseudoaleatorios

Son números generados por un proceso **generador**, estos números **no son totalmente aleatorios**, ya que parten siempre de un valor llamado **semilla.**

Estas definiciones, serán de ayuda para la interpretación que se dará, a lo largo de este trabajo, se irán agregando más definiciones cuando sean requeridas.

SECCIÓN 1.3

Pruebas de aleatoriedad para los números pseudoaleatorios

Recordemos que las variables pueden distinguirse entre discretas y continuas. Por lo que se sabe una variable es discreta cuando no puede tomar ningún valor entre dos números consecutivos, en cambio cuando es una variable continua, si puede tomar cualquier valor dentro de un intervalo.

Es por ellos que lo que se busca es trabajar sobre una distribución uniforme continua, dentro de un intervalo $(0, 1)$, y al saber esto, se sabe que el valor esperado para la media es $\frac{1}{2}$ y el valor esperado para la varianza de $\frac{1}{12}$, esto se puede recalcular partiendo de la función generatriz de momentos.

1.3.1. Prueba de promedios o medias

Una de las propiedades que deben de cumplir los números pseudoaleatorios es que su valor esperado sea $\frac{1}{2}$.

$$H_0 : r_i \sim U(0, 1)$$
$$H_1 : r_i \text{ no están uniformemente distribuidos}$$

Se plantean las hipótesis:

$$H_0 : \mu_{ri} = \frac{1}{2}$$

$$H_1 : \mu_{ri} \neq \frac{1}{2}$$

Nota: ri es una notación para indicar que es un conjunto de números en forma decimal.

La prueba de medias consiste en sumar todos los números generados en su forma decimal y dividirlos entre la cantidad de elementos, para obtener el promedio.

$$\bar{r} = \frac{1}{n}\sum_{i=1}^{n} r_i$$

Una vez que se tengan las hipótesis y la media de los números generados se deben de construir los límites de aceptación o límites de confianza:

Límite Inferior: $$LI\bar{r} = \frac{1}{2} - \left(\frac{z_{\frac{\alpha}{2}}}{\sqrt{12n}}\right)$$

Límite Superior: $$LS\bar{r} = \frac{1}{2} + \left(\frac{z_{\frac{\alpha}{2}}}{\sqrt{12n}}\right)$$

"Si el valor de $\bar{r}$ se encuentra entre los límites de aceptación, concluimos que no se puede rechazar que el conjunto r_i tiene un valor esperado de 0.5 con un nivel de aceptación de $1-\alpha$. En caso contrario se rechaza que el conjunto r_i tiene un valor esperado de 0.5 (ver apéndice D)." [6, pág.35]

Para el cálculo de los límites de aceptación se utiliza el estadístico $z_{\frac{\alpha}{2}}$, el cual se determinó con las tabla de distribución normal estándar.

1.3.2. Prueba de varianzas

Otra propiedad que deben de cumplir los números pseudoaleatorios generados es que su varianza debe de ser $\frac{1}{12}$.

$$H_0 : r_i \sim U(0,1)$$
$$H_1 : r_i \text{ no están uniformemente distribuidos}$$

Para determinar la prueba de varianza, se establecen las siguientes hipótesis:

$$H_0 : {\sigma^2}_{ri} = \frac{1}{12}$$

$$H_1 : {\sigma^2}_{ri} \neq \frac{1}{12}$$

Esta prueba consiste en determinar la varianza de los **n** números pseudoaleatorios mediante la siguiente ecuación:

$$V(r) = \frac{\sum_{i=1}^{n} (r_i - \bar{r})^2}{n-1}$$

Después se calculan los límites de aceptación inferior y superior con las ecuaciones siguientes:

Límite Inferior: $LI_{V(r)} = \dfrac{{\chi^2}_{\frac{\alpha}{2},n-1}}{12\,(n-1)}$

Límite Superior: $LS_{V(r)} = \dfrac{{\chi^2}_{1-\frac{\alpha}{2},n-1}}{12\,(n-1)}$

Si el valor de $V(r)$ se encuentra entre los límites de aceptación, decimos que no se puede rechazar que el conjunto tiene una varianza de $\frac{1}{12}$, con un nivel de aceptación de $(1-\alpha)$; de lo contrario se rechaza que el conjunto tiene una varianza de $\frac{1}{12}$.

1.3.3. Prueba de χ^2

Las propiedad más importante que debe cumplir un conjunto de números r_i es la de uniformidad. Para su comprobación se han desarrollado pruebas estadísticas con χ. Para lo anterior es necesario formular las siguientes hipótesis:

$$H_0 : r_i \sim U(0,1)$$
$$H_1 : r_i \text{ no están uniformemente distribuidos}$$

La prueba χ busca determinar cuándo el conjunto de números pseudoaleatorios se distribuye de una manera uniforme dentro del intervalo $(0,1)$.

Para poder hacer esta prueba se divide el intervalo en **m** subintervalos de la forma $m = \sqrt{n}$, donde n es el total de números generados, a lo anterior se le llama la regla de **Velleman**, donde se considera $(50 < n < 100)$, de lo contrario debe de ocuparse la regla de **Sturges** donde la muestra es considerablemente grande, para garantizar su confiabilidad. [10].

La expresión es:

$$k = 1 + \left(\frac{10}{3}\right) Log_{10}(n)$$

Donde:

- k representa el número de intervalos, el cual debe ser un número entero, de no serlo se redondea al entero más cercano, si es mayor o igual al 0.5 se redondea hacia arriba y si es menor o igual a 0.4 se redondea al entero dado.
- n representa la cantidad de números pseudoaleatorios generados.

Tenemos dos tipos de frecuencias:

- La frecuencia **observada** se representa como O_i que es la cantidad de números pseudoaleatorios (r_i) que se clasifican en cada intervalo.
- La otra frecuencia es la **esperada** que se expresa como E_i es la cantidad de números pseudoaleatorios (r_i) que se espera encontrar en cada intervalo.

A partir de los valores O_i y de E_i se calcula la frecuencia esperada con $E_i = \frac{n}{m}$, mientras que la frecuencia observada son cuantos números están en cada intervalo, para después determinar el estadístico ${\chi_0}^2$ mediante la siguiente ecuación:

$${\chi_0}^2 = \sum_{I=1}^{n} \frac{(E_i - \quad O_i)^2}{E_i}$$

Si el valor estadístico ${\chi_0}^2$ es menor al valor de las tablas de ${\chi^2}_{\alpha,n-1}$, no se puede rechazar que el conjunto de números r_i sigue una distrbución uniforme. En caso contrario, se rechaza.

A continuación se estudiarán diversos algoritmos para generar números pseudoaleatorios de cuatro dígitos, posteriormente se presenta la generalización de estos para ocho dígitos.

CAPÍTULO 2

TIPOS DE GENERADORES

SECCIÓN 2.1

Generadores de números pseudoaleatorios de 4 dígitos

Se presentan los **principales generadores** de números pseudoaleatorios, están divididos en dos grupos: congruenciales y no congruenciales, que estos últimos a su vez se dividen en lineales y no lineales, como se muestra en la Figura 2.1.

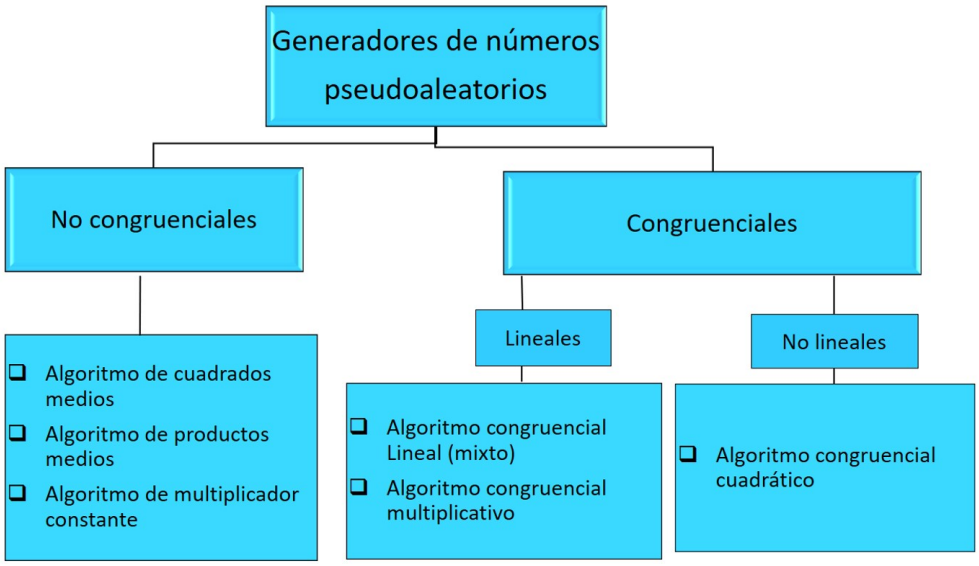


Figura 2.1: Clasificación de los generadores

2.1.1. Generadores no congruenciales

2.1.1.1. Algoritmo de cuadrados medios

Este algoritmo se propuso en la década de los cuarenta por Von Neumann y Metrópolis, este generador consiste en lo siguiente:

1. Se propone una semilla para empezar a generar los números, la cantidad de dígitos que debe tener son cuatro.
2. La semilla se nombra como x_0
3. La semilla se eleva al cuadrado y del número obtenido se toman los cuatro dígitos del centro.
4. Se actualiza la semilla y se repite el paso anterior.

Nota: los números obtenidos deberán estar dentro del intervalo (0,1), como ya se vio en el Capitulo 1, partimos de que son uniformemente distribuidos y provienen de una función continua, estos se presentarán de forma decimal para verificar que sí se encuentran dentro de dicho intervalo, tanto para este generador como para los demás generadores que se presentaran más adelante.

Ejemplo:

Con el algoritmo de cuadrados medios se generan números pseudoaleatorios de cuatro dígitos y se representa de la siguiente manera:

La semilla que se propone es 2158 y se generan los números, como lo menciona el algoritmo en el paso 3.

Se tomarán los dígitos del centro, cuando no sea posible obtener los dígitos del centro, porque da como resultado un número con una cantidad impar de dígitos y no es posible dividirlo en dos partes iguales, se agregará un cero al lado izquierdo del número obtenido, como se muestra a continuación.

$$(2158)^2 \quad = \quad 04\underline{6569}64 \quad \Rightarrow \quad x_1 = 6569$$

Los r_i se calculan como $r_i = \dfrac{x_i}{10000}$ con $i = 0, 1, 2, ...$

$$(2158)^2 = 04\underline{6569}64 \Rightarrow x_1 = 6569 \Rightarrow r_1 = 0.6569$$

$$(6569)^2 = 43\underline{1517}61 \Rightarrow x_2 = 1517 \Rightarrow r_2 = 0.1517$$

$$(1517)^2 = 02\underline{3012}89 \Rightarrow x_3 = 3012 \Rightarrow r_3 = 0.3012$$

$$(3012)^2 = 09\underline{0721}44 \Rightarrow x_4 = 0721 \Rightarrow r_4 = 0.0721$$

Como se puede observar, con este algoritmo es muy fácil generar números pseudoaleatorios, para no calcularlos a mano se puede implementar el algoritmo en un software y así poder generar los que sean necesarios. El software que se usará es **Java**, en él se realizará la implementación con la misma semilla utilizada, para verificar que sí genera los mismos números pseudoaleatorios calculados.

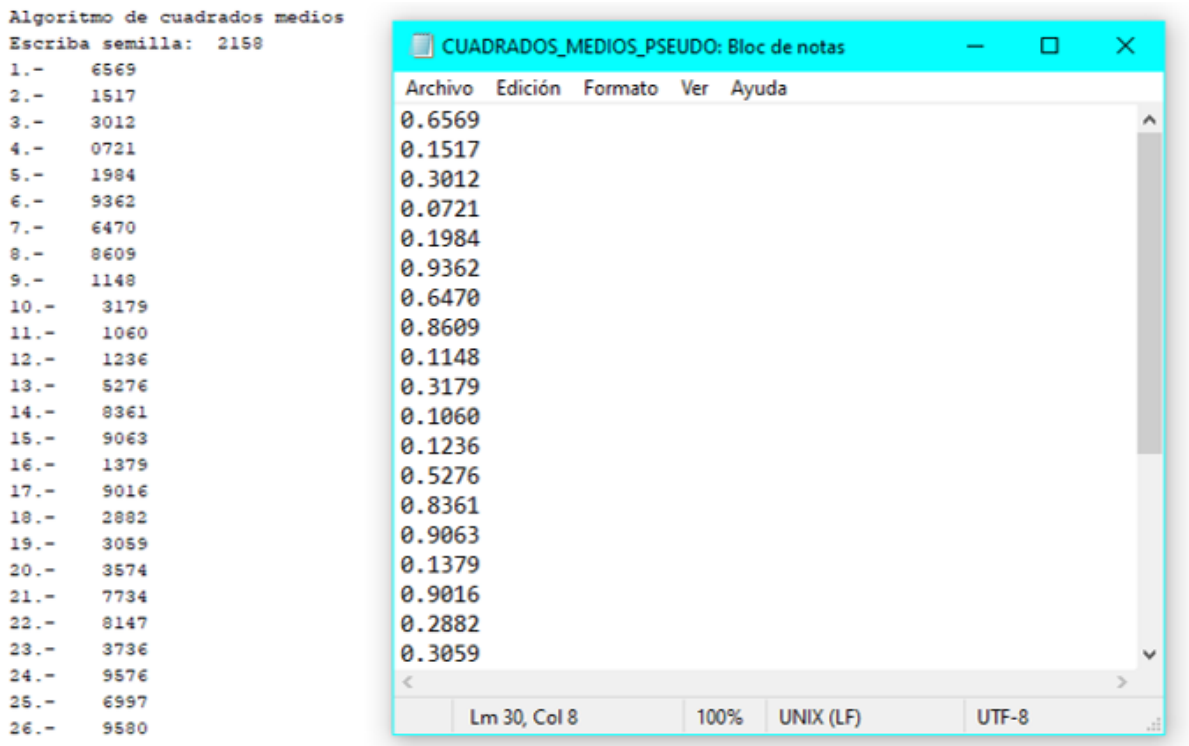


Figura 2.2: Generación de números pseudoaleatorios con algoritmo de cuadrados medios con una semilla $x_0 = 2158$.

Este primer ejemplo genera 30 números pseudoaleatorios, a partir de la semilla propuesta, los r_i son los números en forma decimal, los cuales siempre estarán dentro del intervalo (0,1), estos son guardados en un archivo para después hacer un análisis en las pruebas de aleatoriedad.

Ejemplo: ¿Cuándo el algoritmo no genera números?

El algoritmo es incapaz de generar una secuencia de números con la semilla propuesta, aquí es cuando encontramos que hay problemas y no siempre es posible generar números pseudoaleatorios.

Por ejemplo si $x_0 = 1000$, al momento de intentar encontrar una nueva semilla se tiene que:

$$(1000)^2 \quad = \quad 01\underline{000}00 \quad \Rightarrow \quad x_1 = 0000$$

Se concluye que este generador no funciona con esta semilla, por eso debemos definir bien la semilla con la que se va a trabajar. Se debe de tener presente que no cualquier tipo de semilla se puede utilizar, ya que como se pudo notar en este ejemplo, al tener un número con 3 de sus dígitos ceros y al elevarlo al cuadrado se presenta la situación mostrada. Por lo anterior debemos procurar que la semilla que se proponga no tenga 3 ceros consecutivos en la parte final de la semilla.

2.1.1.2. Algoritmo de productos medios

La generación de números pseudoaleatorios, por medio de este algoritmo es similar al de cuadrados medios, sólo que para éste se necesitan dos semillas.

1. Semilla 1, x_0, número con cuatro dígitos
2. Semilla 2, x_1, número con cuatro dígitos
3. Se multiplican las dos semillas, del resultado se seleccionan los cuatro dígitos del centro.
4. Si no es posible obtener los dígitos del centro porque se tiene un número (impar) de dígitos, se agrega un cero a la izquierda del número obtenido.

Ejemplo:

Se propone dos semillas $x_0 = 2158$ y $x_1 = 1246$ se procede de acuerdo al algoritmo

$$(2158)(1246) \quad = \quad 02\underline{688}868 \quad \Rightarrow \quad x_2 = 6888$$

los r_i son números que deben de estar dentro del intervalo (0,1), estos se calculan así $r_i = \dfrac{x_i}{10000}$ con $i = 0, 1, 2, ...$

$$(2158)(1246) \quad = \quad 02\underline{688}868 \quad \Rightarrow \quad x_2 = 6888 \quad \Rightarrow \quad r_2 = 0.6888$$

$$(1246)(6888) \quad = \quad 08\underline{582}448 \quad \Rightarrow \quad x_3 = 5824 \quad \Rightarrow \quad r_3 = 0.5824$$

$$(6888)(5824) \quad = \quad 40\underline{115}712 \quad \Rightarrow \quad x_4 = 1157 \quad \Rightarrow \quad r_4 = 0.1157$$

$$(5824)(1157) \quad = \quad 06\underline{738}368 \quad \Rightarrow \quad x_5 = 7383 \quad \Rightarrow \quad r_5 = 0.7383$$

Este generador también se programó en **Java** y genera los mismos números pseudoaleatorios calculados.

```
Algoritmo de productos medios
Escribe la semilla1:  2158
Escribe la semilla2:  1246
1.-    6888
2.-    5824
3.-    1157
4.-    7383
5.-    5421
6.-    0232
7.-    2576
8.-    9763
9.-    1494
10.-    5859
11.-    7533
12.-    1358
13.-    2298
14.-    1206
15.-    7713
16.-    3018
17.-    2778
18.-    3840
19.-    6675
20.-    6320
21.-    1860
22.-    7552
23.-    0467
24.-    5267
25.-    4596
```

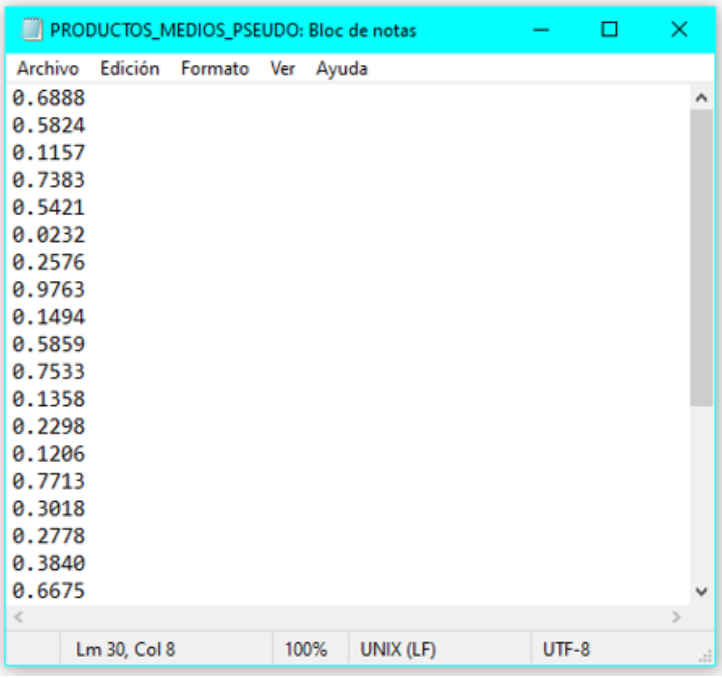


Figura 2.3: Generación de números pseudoaleatorios con el algoritmo de productos medios, con dos semillas $x_0 = 2158$ y $x_1 = 1246$

Como se puede observar se generan números pseudoaleatorios de una manera muy fácil y rápida, sólo es necesario elegir las semillas de forma tal que proporcionen un periodo de vida largo.

2.1.1.3. Algoritmo de multiplicador constante

Este generador necesita una semilla y una constante para poder generar los números pseudoaleatorios.

1. La semilla es x_0, un número de cuatro dígitos
2. La constante a un número de cuatro dígitos

   Nota: De preferencia la semilla y la constante deben ser del mismo tamaño (cuatro dígitos).
3. Este generador multiplica la constante por la semilla y la semilla es la que debe actualizarse, es decir una vez que se multiplique x_i y a, se obtendrá la actualización de la semilla x_i.
4. Si no es posible obtener los dígitos del centro por que se tiene un número con una cantidad impar de dígitos, se agrega un cero a la izquierda para que, al hacer la división de la cantidad de dígitos en dos partes iguales se obtengan los cuatro dígitos del centro.

Ejemplo:

Se propone una constante $a = 2158$, y una semilla $x_0 = 1246$ y se generan los números.

$$(2158)(1246) \quad = \quad 02\underline{6888}68 \quad \Rightarrow \quad x_1 = 6888$$

Los r_i se calculan mediante la formula $r_i = \dfrac{x_i}{10000}$ con $i = 0, 1, 2, ...$

$$(2158)(1246) \quad = \quad 02\underline{6888}68 \quad \Rightarrow \quad x_1 = 6888 \quad \Rightarrow \quad r_1 = 0.6888$$

$$(2158)(6888) \quad = \quad 14\underline{8643}04 \quad \Rightarrow \quad x_2 = 8643 \quad \Rightarrow \quad r_2 = 0.8643$$

$$(2158)(8643) \quad = \quad 18\underline{6515}94 \quad \Rightarrow \quad x_3 = 6515 \quad \Rightarrow \quad r_3 = 0.6515$$

$$(2158)(6515) = 14\underline{0593}70 \Rightarrow x_4 = 0593 \Rightarrow r_4 = 0.0593$$

$$(2158)(0593) = 01\underline{2796}94 \Rightarrow x_5 = 2796 \Rightarrow r_5 = 0.2796$$

Se muestra la ejecución del programa con la semilla y constante dadas, recordando que el algoritmo se realizó en **Java**.

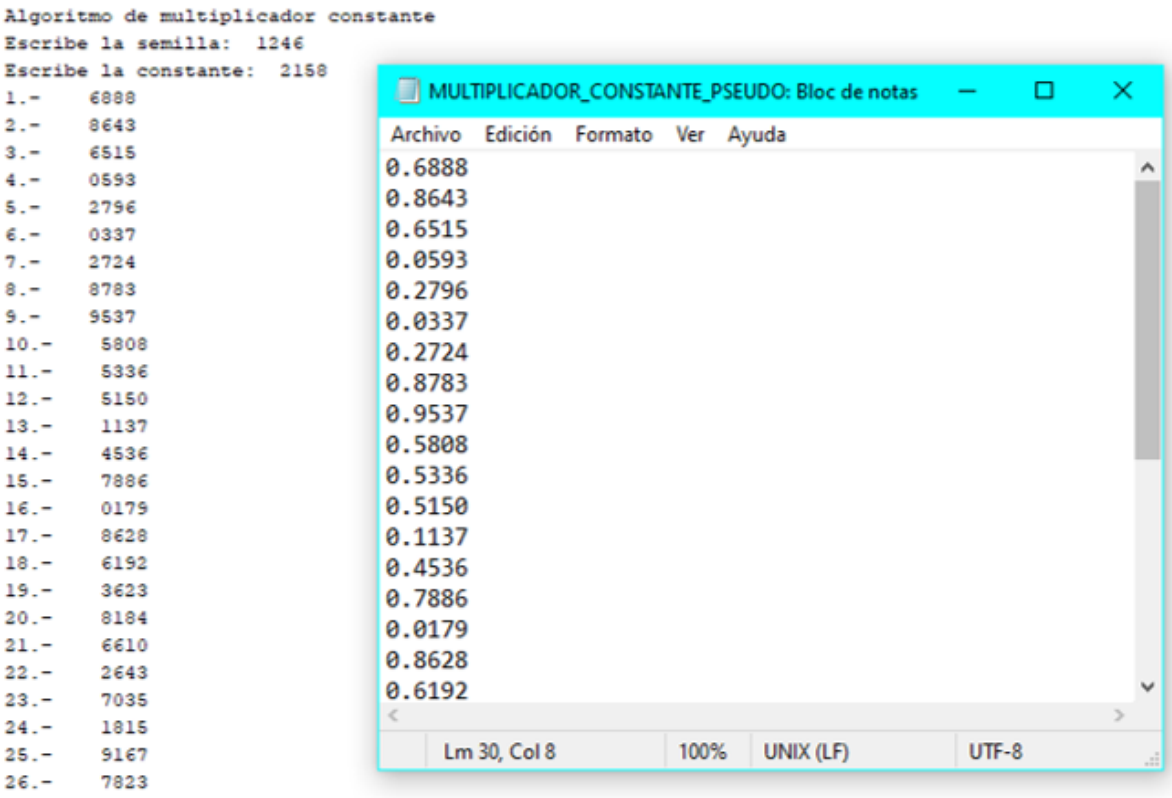


Figura 2.4: Simulación del generador multiplicador constante, con a=2158 y $x_0 = 1246$

Se realizaron simulaciones con un número de iteraciones grande, además de ingresar semillas y constantes diferentes, lo anterior con el fin de determinar la mejor semilla en cuanto a la cantidad de números pseudoaleatorios que se generan.

2.1.2. Generadores congruenciales lineales

"Estos números se consideran pseudoaleatorios, porque aunque pasan todas las pruebas estadísticas de aleatoriedad, que se mencionaron en el Capítulo 1, Sección 1.3, ellos son completamente determinísticos.

Actualmente, casi todas las computadoras incluyen en sus programas de biblioteca alguna variante de los métodos congruenciales sugeridos por **Lehmer**. Los dos métodos congruenciales más populares son: congruencial mixto y congruencial multiplicativo." [4, pág.20]

"Lehmer propuso el generador lineal de congruencial, el cual con pequeñas modificaciones propuestas por **Thomson y Rotenberg**, ha llegado a convertirse en el método para la generación de números pseudoaleatorios más ampliamente usado en la actualidad." [2, pág.51]

Antes de iniciar este algoritmo hablemos de la siguiente notación:

$$a \equiv b \, mod \, m$$

Esto quiere decir que **a** es congruente con **b** modulo **m** o decimos que **m** divide a **(a-b)**. Si dividimos al número **a** entre **m** y al número **b** entre **m** nos quedará el mismo residuo. Por ejemplo:

$$3 \equiv 17 \, mod \, 2$$

por que el 2 divide al $3 - 17 = -14$ y si dividimos el 3 entre 2 da como modulo 1 y si dividimos el 17 entre el 2 también da como residuo 1.

Una vez que se sabe como trabaja el **mod**, entonces podemos hacer la siguiente observación:

Todo número entero positivo **z** cumple solo una de las siguientes afirmaciones.

$$\begin{aligned} 0 &\equiv z \, mod \, m \\ 1 &\equiv z \, mod \, m \\ 2 &\equiv z \, mod \, m \\ 3 &\equiv z \, mod \, m \\ &\vdots \\ m-1 &\equiv z \, mod \, m \end{aligned}$$

donde **m** es un número entero positivo fijo.

2.1.2.1. Algoritmo congruencial lineal (mixto)

"Los generadores congruenciales lineales generan una secuencia de números pseudoaleatorios en la cual el próximo número pseudoaleatorio es determinado a partir del último número generado, es decir, el número pseudoaleatorio x_{i+1} es derivado a partir del número x_i." [6, pág.27], para este generador se necesita que los números propuestos sean positivos y mayores a cero.

1. Semilla $x_0 > 0$
2. Constante multiplicativa $a > 0$
3. Constante aditiva $c > 0$
4. Módulo $m > 0$

$$x_{i+1} = ax_i + c \, mod \, m \qquad i = 1, 2, 3, ... \qquad r_i = \frac{x_i}{m-1}$$

Nota: se debe de recordar que los r_i en los algoritmos no congruenciales son de la forma $r_i = \frac{x_i}{10000}$, es este tipo de generadores los r_i se denotarán con:

$$r_i = \frac{x_i}{m-1}$$

Se deben de seleccionar de forma adecuada los parámetros para que tengan un periodo grande, para ello se deben de tomar en cuenta las siguientes condiciones:

"**m** debe ser múltiplo de 2^g, donde g será un entero, además $a = 1 + 4k$, con k un entero, x_0 también es un número entero y c debe ser primo relativo a m" [6, pág.27]

Ejemplo:

Se proponen los siguientes parámetros para el generador, $x_0 = 2158$ de tamaño cuatro, para generar números de cuatro cifras, $a = 61$, $c = 1$, $m = 9990$. Los utilizamos directamente en el algoritmo ya implementado en **Java**.

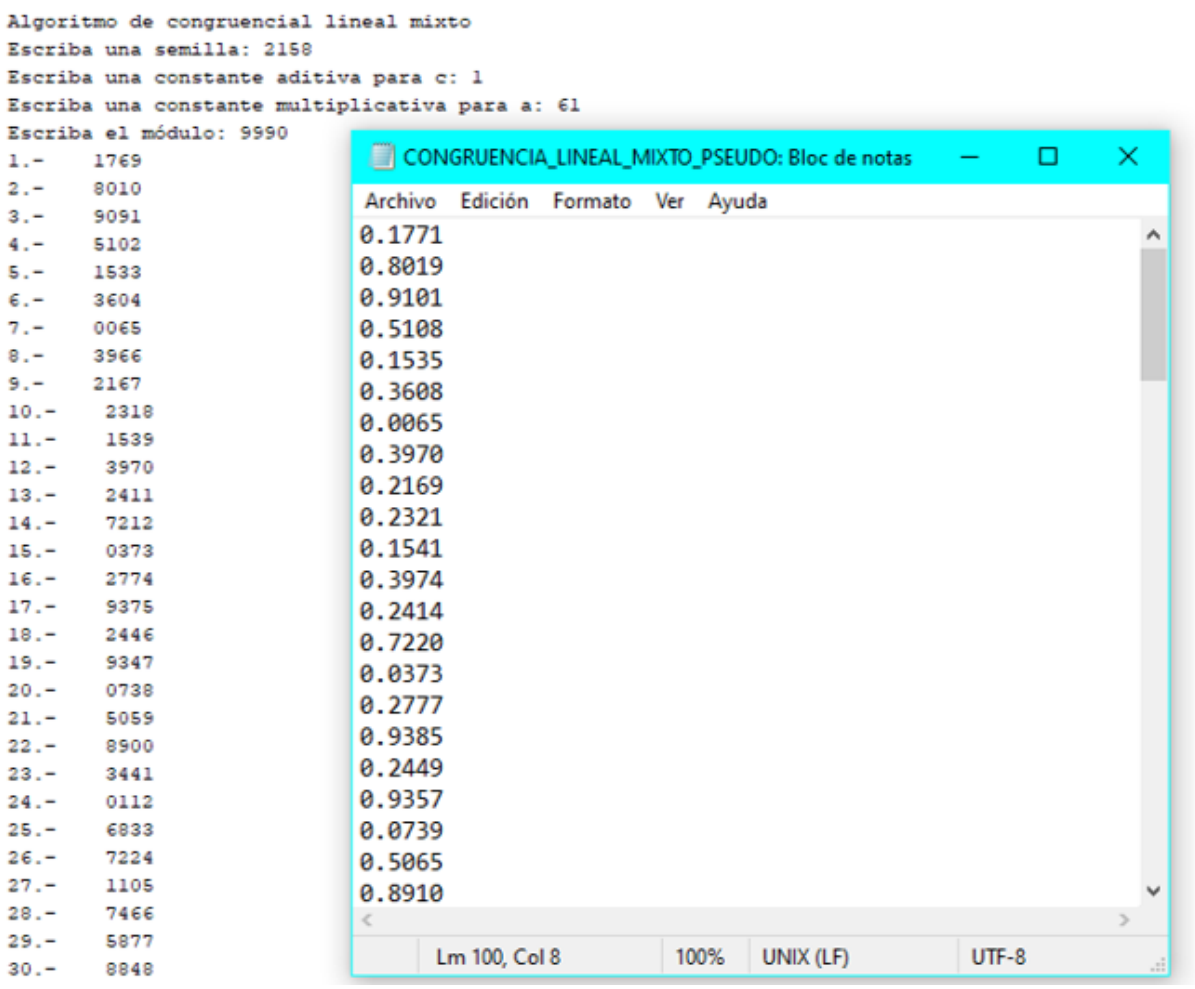


Figura 2.5: Simulación del algoritmo congruencial lineal mixto, con periodo de vida $m = 9990$

Si comparamos los números generados con los números en su expresión decimal del bloc de notas observamos que no es inmediata su identificación, pues estos se dividen por $m-1$ y no por 10000, ésta es una diferencia que se encuentra en este algoritmo que no se encuentra en los algoritmos no congruenciales.

Como se puede observar en el ejemplo anterior, m representa un periodo de vida, después de esa cantidad de números generados, los números pseudoaleatorios se empiezan a repetir.

De la misma forma se puede ver en la Figura 2.5 que se generaron los r_i, los utilizaremos para hacer el análisis y pruebas de aleatoriedad posteriormente.

2.1.2.2. Algoritmo congruencial multiplicativo

Este algoritmo surge del algoritmo lineal mixto, cuando $c = 0$ por lo que se necesita, sólo tener un número para el modulo, una semilla y una constante multiplicativa, se aplica el algoritmo y se obtienen los números pseudoaleatorios.

1. La semilla es x_0 un número mayor que cero

2. La constante multiplicativa es a mayor que cero

3. El módulo es m mayor que cero

$$x_{i+1} = ax_i \, mod \, m \qquad i = 1, 2, 3, ... \qquad r_i = \frac{x_i}{m-1}$$

"Las **condiciones** que deben de cumplir los números involucrados para que tenga un periodo de vida completo deben de ser: **m** múltiplo de 2^g, donde g es un entero, $a = 3+8k$, con $k = 0, 1, 2, 3, ...$ y x_0 un número impar." [6, pág.29]

Ejemplo:

Para este generador se proponen parámetros totalmente diferentes al generador anterior y son los siguientes, la semilla $x_0 = 211$, la constante multiplicativa $a = 123$ y el módulo $m = 9984$ para realizar la simulación como se ve en la Figura 2.6.

Siempre se debe elegir un módulo grande y que cumpla las condiciones dadas para este generador y así poder garantizar que el algoritmo sea eficaz.

Lo que buscamos en un generador es que tenga un periodo de vida largo, sea rápido y eficaz, más adelante se hará el análisis y la comparación entre los diferentes algoritmos generadores.

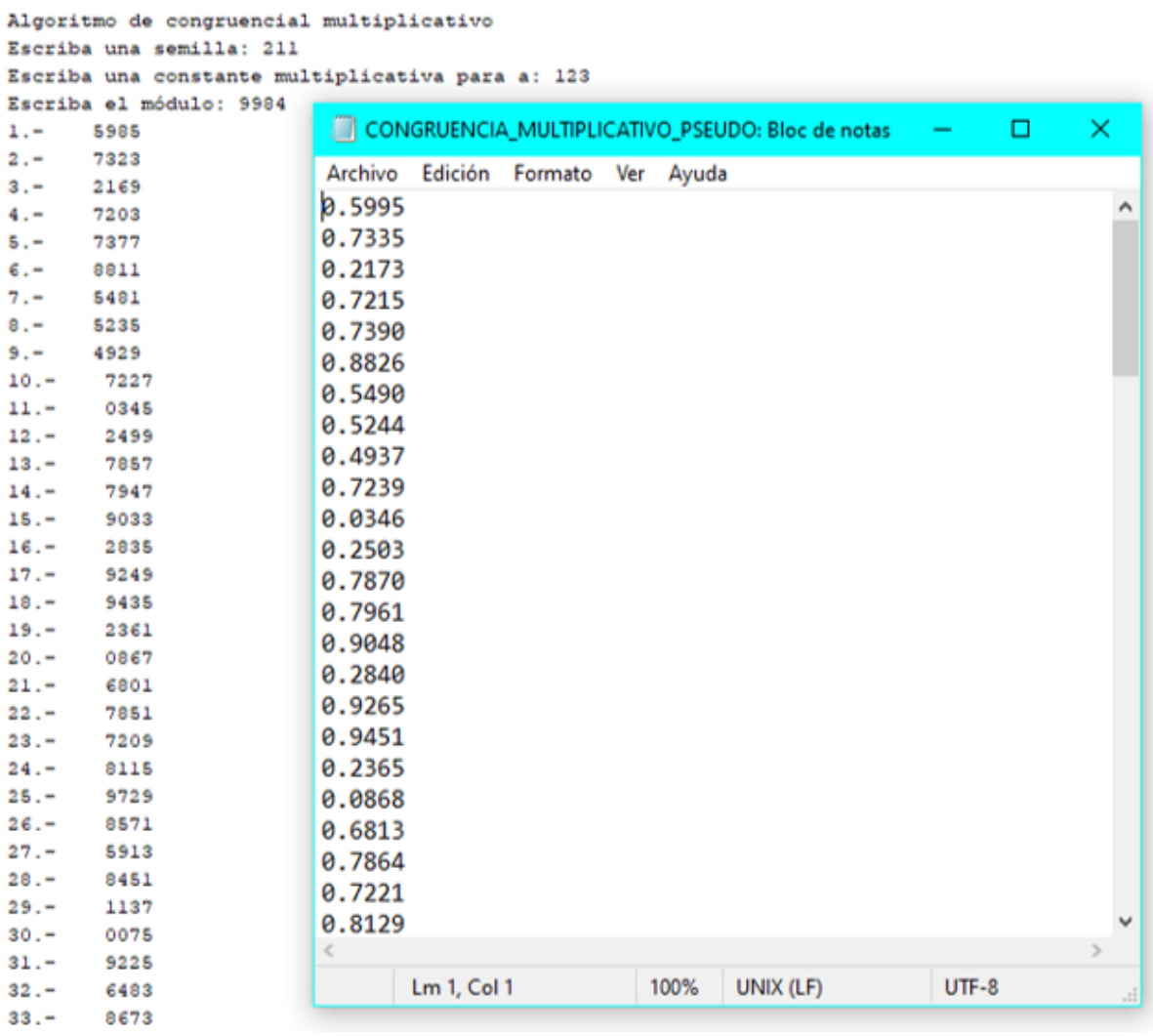


Figura 2.6: Simulación de congruencial multiplicativo, con periodo $m = 9984$

2.1.3. Generadores congruenciales no lineales

Dentro de los algoritmos congruenciales no lineales están el algoritmo congruencial cuadrático y el de Blum Blum.

2.1.3.1. Algoritmo congruencial cuadrático

Para este generador se necesita que los números propuestos sean positivos y mayores a cero.

1. Semilla $x_0 > 0$

2. Constante multiplicativa del término cuadrático $a > 0$

3. Constante aditiva $c > 0$

4. Constante multiplicativa $b > 0$

5. Módulo $m > 0$

   La ecuación recursiva que se utiliza para la generación de los números pseudoaleatorios por medio de este algoritmo es:

$$x_{i+1} = a{x_i}^2 + bx_i + c \, mod \, m \qquad i = 1, 2, 3, ... \quad \text{y} \quad r_i = \frac{x_i}{m-1}$$

"Las condiciones que deben de cumplir los parámetros **m, a, b y c** para alcanzar un periodo máximo son:" [6, pág.31]

- $m = 2^g$
- g un entero
- a un número par
- c un número impar
- $b - 1 \, mod \, 4 = 1$

Ejemplo:

Se proponen los siguientes valores para los parámetros, $x_0 = 2158$, $a = 22$, $b = 50$, $c = 17$, $g = 32$, $m = 9984$, se obtienen los siguientes números pseudoaleatorios, donde los r_i, también son de la forma $r_i = \frac{x_i}{m-1}$

Al hacer la simulación con este generador vemos que es posible tener un gran número de números pseudoaleatorios. Pero no perdamos el objetivo principal que es la generación de estos números pseudoaleatorios pero de **ocho dígitos**.

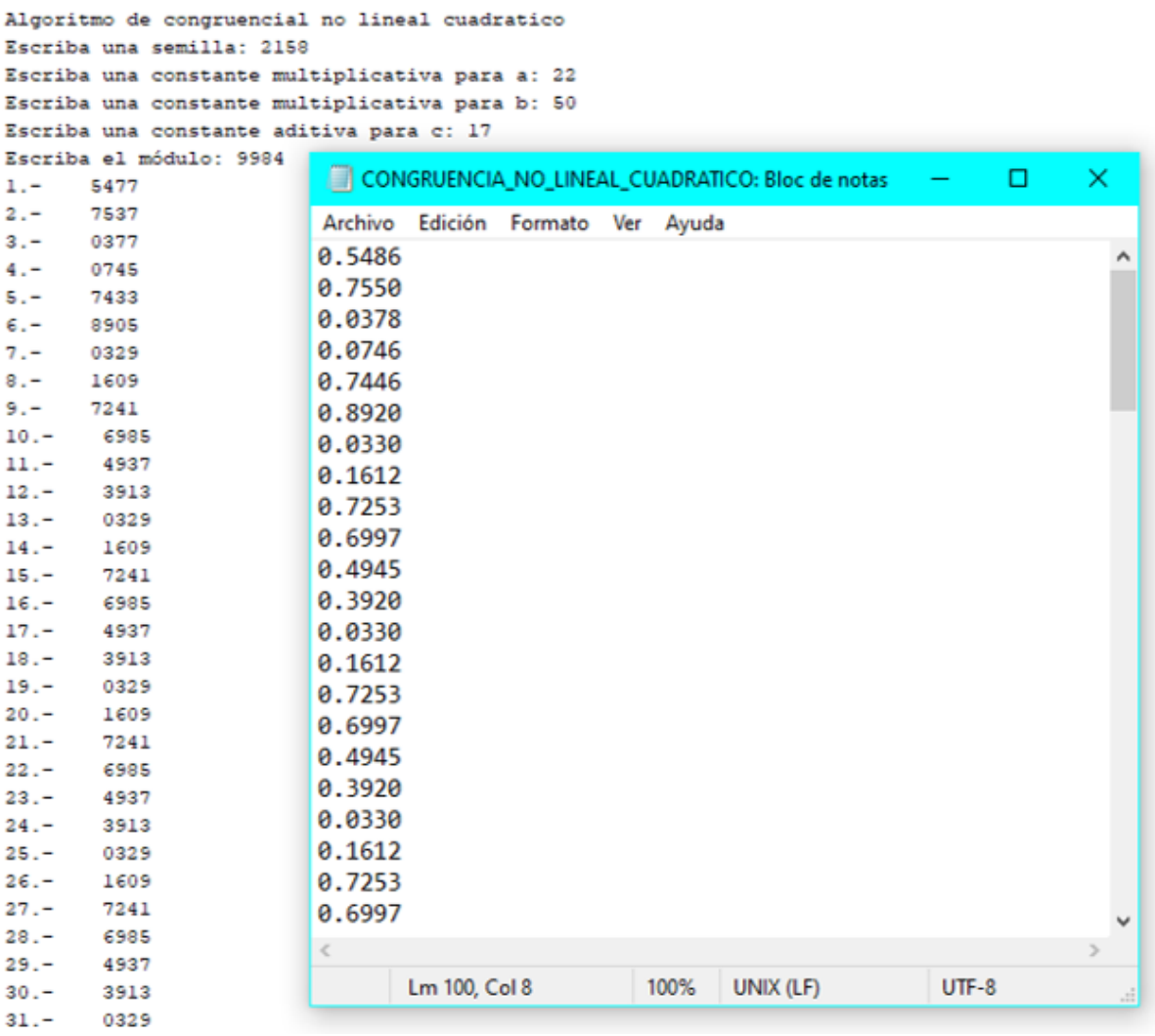


Figura 2.7: Simulación del algoritmo congruencia cuadrático, con un periodo de vida de 9984.

Se realizará el mismo proceso anterior con cada generador y después se analizará cuál de todos los algoritmos genera la cantidad máxima de números pseudoaleatorios, además de que cumplan las pruebas de aleatoriedad que se explicaron en el capítulo 1.

Una vez hecho este análisis entre los generadores, escogeremos el que cumpla con las características necesarias para trabajar con él y así tomar la decisión para hacer nuestro token.

SECCIÓN 2.2

Generadores de números pseudoaleatorios de ocho dígitos

En la sección anterior se presentaron los algoritmos para cuatro dígitos, esos fueron modificados con las condiciones necesarias y ahora generan los números pseudoaleatorios de ocho dígitos. Lo que corresponde ahora es analizar cada generador con sus parámetros junto con sus pruebas de aleatoriedad de cada uno.

2.2.1. Cuadrados medios

El primer generador que se analiza es el de **cuadrados medios**, este generador tiene un periodo de vida de $10,000$, pero ¿comó sabemos esto?

Para poder saber el periodo no hay alguna fórmula que se use directamente, hay que realizarlo por **ensayo y error**. La primera vez que se ejecutó el programa, en **Java**, se generaron $5,000,000$ números pseudoaleatorios y el programa se ejecuto exitosamente.

A esos números generados se les aplicó las tres pruebas de aleatoriedad y ninguna fue satisfactoria. Se repite el proceso descrito, muchas veces hasta llegar a el número $10,000$. **Nota**: la semilla a utilizar consta de ocho dígitos esto es la clave principal.

El código se modificó y se incluyó en el Apéndice B, por si se desea consultar. Los r_i obtenidos se guardaron en un archivo de texto para realizarle las pruebas de aleatoriedad en R.

A continuación se presenta la simulación del generador con la semilla $x_0 = 38765294$, en la Figura 2.8 se presenta la lista de los primeros 24 números pseudoaleatorios generados y en el archivo de texto se observan los primeros 15 números.

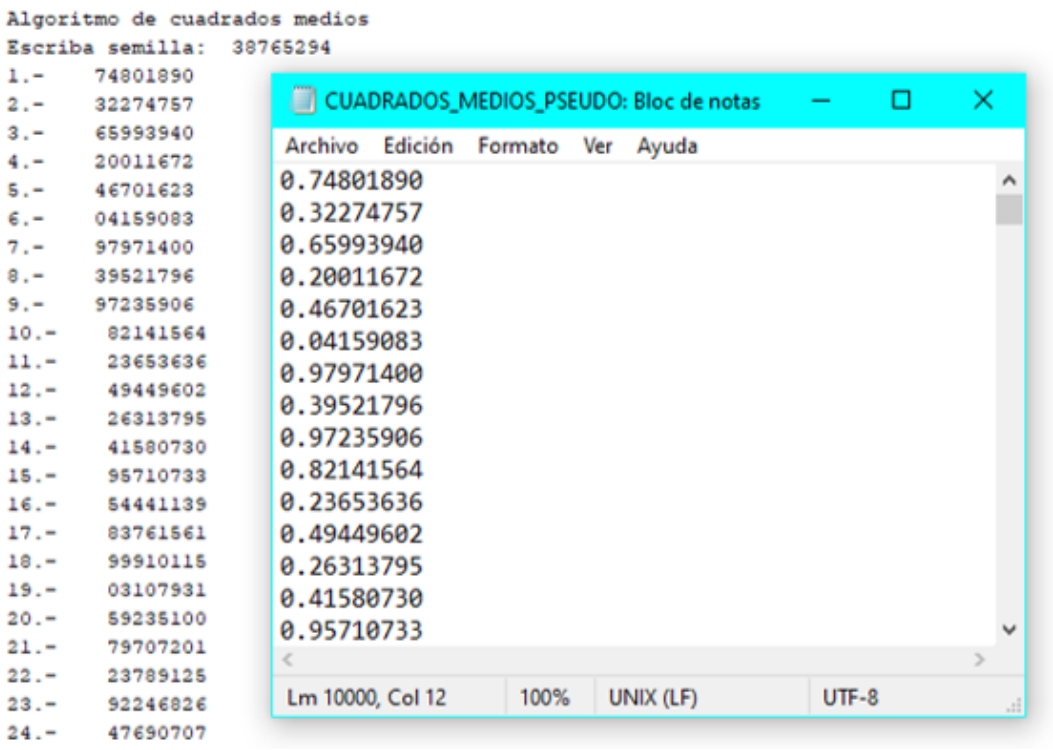


Figura 2.8: Inicio de la lista de números pseudoaleatoios por el algoritmo de cuadrados medios con la semilla $x_0 = 38765294$

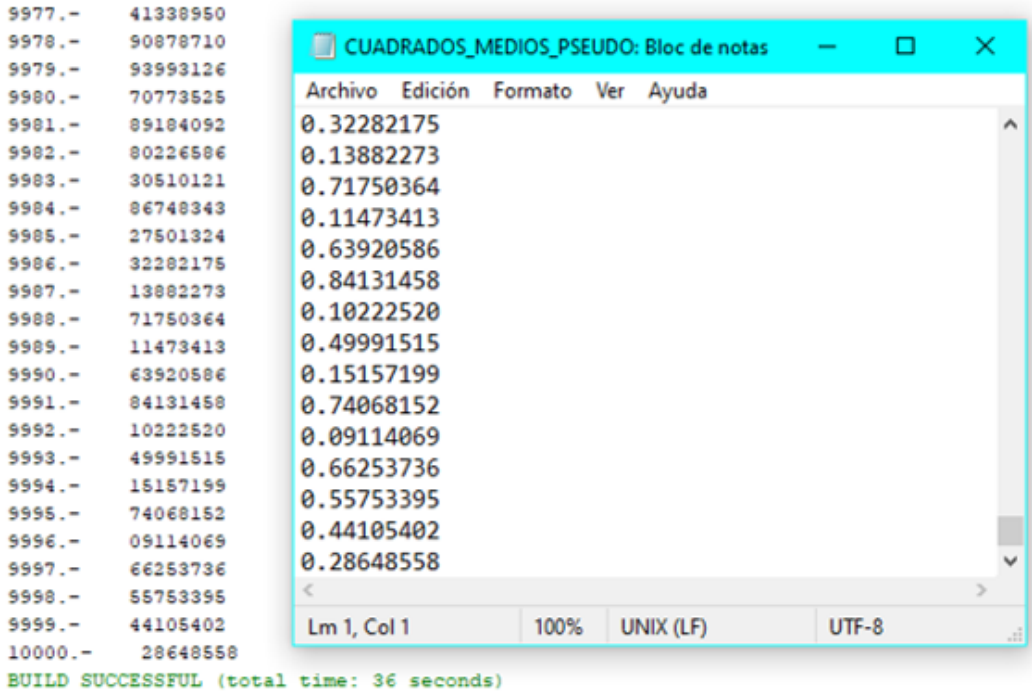


Figura 2.9: Final de la lista de números pseudoaleatorios por el algoritmo de cuadrados medios con un periodo de $10,000$

Una vez que tenemos los r_i asociados a los números pseudoaleatorios en el bloc de notas, utilizamos el software llamado **R** para realizarle las pruebas de aleatoriedad.

Este código se usa para todos los generadores sólo se cambia la linea donde se manda a llamar al archivo que contiene los datos de los r_i asociadas a cada generador.

Prueba de medias

```
> # PRUEBA DE MEDIA
> # Se acepta la hipotesis si la media de los pseudoaleatorios esta dentro del intervalo
> # [lim.inf.pseudo , lim.sup.pseudo] o es igual a 0.5
> # si no se rechaza
>
> pseudoaleatorios<-scan("D:/Users/Anita Ramirez/Documents/DESARROLLANDO TESIS/PROGRAMAS
 8 DIGITOS/NO CONGRUENCIALES/cuadradosmedios/CUADRADOS_MEDIOS_PSEUDO.txt")
Read 10000 items
>
> alfa=0.05;
> z<-abs(qnorm(alfa/2))
> z
[1] 1.959964
>
> # Intervalos
> lim.inf.pseudo<-0.5-(z/sqrt(12*n))
> lim.inf.pseudo
[1] 0.4943421
>
> lim.sup.pseudo<-0.5+(z/sqrt(12*n))
> lim.sup.pseudo
[1] 0.5056579
>
> # Media de los pseudoaleatorios
> media.pseudo<-mean(pseudoaleatorios)
> media.pseudo
[1] 0.4975468
>
```

Figura 2.10: Resultados de la prueba de medias

En la Figura 2.10, en la linea donde dice:

`pseudoaleatorios< −scan(“D:/Users/Anita Ramirez/Documents/DESARROLLO TESIS/PROGRAMAS 8 DIGITOS/NO CONGRUENCIALES/CUADRADOSMEDIOS/CUADRADOS_MEDIOS-_PSEUDO.txt”)`

Esta linea es la única que se modificará para llamar el archivo de texto donde se guardarán los r_i asociados a los números pseudoaleatorios de cada generador.

Analizando los resultados obtenidos en la prueba de medias, que se puede observar en la Figura 2.10, se propone una hipótesis que dice: "se acepta la hipótesis si la media de los números pseudoaleatorios cae dentro del intervalo de confianza ó es igual a $\frac{1}{2}$".

¿Qué quiere decir esto?

Anteriormente se menciono que el valor esperado para la media de una variable uniforme es $\frac{1}{2}$, y si estamos asegurando que este conjunto de datos, se comportan como una distribución uniforme, entonces debe cumplir esta hipótesis, en caso contrario que no sea igual, deberá estar dentro del intervalo propuesto, recordemos que el intervalo de confianza no es una estimación puntual, sino una estimación aproximada para encontrar la media de la población, siendo acotada entre dos valores en donde se encontrará la media de este conjunto de datos.

Prueba de varianza

Como se puede observar la Figura 2.11, la prueba de varianza lo que hace es proponer un intervalo acotado, para poder encontrar una estimación de la varianza de los números pseudoaleatorios ó que sea igual a $\frac{1}{12}$, sabiendo que provienen de una distribución uniforme, sobre el intervalo (0,1) y así poder aceptar la hipótesis.

```
> # PRUEBA DE VARIANZA
> # Se acepta la hipotesis si la varianza de los pseudoaleatorios esta dentro
> # del intervalo [lim.inf.var , lim.sup.var] o es igual a 1/12
> # si no se rechaza
>
> #Intervalos
> lim.inf.var<-(chi.inf/(12*n-1))
> lim.inf.var
[1] 0.08103178
>
> lim.sup.var<-(chi.sup/(12*n-1))
> lim.sup.var
[1] 0.08565118
>
> #varianza de lospseudoaleatorios
> var(pseudoaleatorios)
[1] 0.08226933
>
```

Figura 2.11: Resultados de la prueba de varianza

Prueba de χ^2

En esta prueba lo que se hace es confirmar que los números pseudoaleatorios vienen de una distribución uniforme, por lo que se hace una tabla de frecuencias, como se muestra en la Tabla 2.1.

Se observa que son 15 clases con sus intervalos correspondientes, pero ¿como se sabe que son 15?, para calcular el número de clases o marca de clase se usa la fórmula de Sturges, que se mencionó en el Capítulo 1 y después se calcula los intervalos, para poder armar la tabla de frecuencias.

Los **Oi** son la frecuencia observada en cada intervalo, así se sabe cuantos datos caen en los intervalos creados, continuamos con los **Ei** que es la frecuencia esperada para cada intervalo ya que se espera que sea la misma en todos y por último **ecu** representa la operación $\frac{(E_i - O_i)^2}{2}$, la cual se hace por filas, todo esto es necesario para obtener el estadístico de los números pseudoaleatorios generados, y poder hacer una comparación con el estadístico de las tablas de χ^2.

Toda esta información se obtiene del archivo .txt, donde se almacenan los números pseudoaleatorios, que para este ejemplo los obtenemos de la Figura 2.8 y 2.9, del bloc de notas, y así poder tener la Tabla 2.1.

	tabla.intervalos	Oi	Ei	ecu
1	(-0.000893,0.0668]	680	666.6667	0.266666667
2	(0.0668,0.133]	649	666.6667	0.468166667
3	(0.133,0.2]	648	666.6667	0.522666667
4	(0.2,0.267]	662	666.6667	0.032666667
5	(0.267,0.333]	665	666.6667	0.004166667
6	(0.333,0.4]	710	666.6667	2.816666667
7	(0.4,0.467]	690	666.6667	0.816666667
8	(0.467,0.533]	672	666.6667	0.042666667
9	(0.533,0.6]	701	666.6667	1.768166667
10	(0.6,0.667]	661	666.6667	0.048166667
11	(0.667,0.733]	644	666.6667	0.770666667
12	(0.733,0.8]	655	666.6667	0.204166667
13	(0.8,0.867]	659	666.6667	0.088166667
14	(0.867,0.933]	679	666.6667	0.228166667
15	(0.933,1]	625	666.6667	2.604166667

Tabla 2.1: Tabla de frecuencias para la prueba de χ^2 para el generador cuadrados medios.

```
> # PRUEBA DE LA JI-CUADRADA
> # El estadistico x_0 debe ser menor al estadistico correspondientede de (x0^2)_(alfa,n-1)
> # si el estadistico es mayor al (x0^2)_(alfa,n-1) entonces se rechaza.
>
> # Estadistico (x0^2)_(alfa,n-1)
> chi.tabla<-qchisq(alfa,k1-1,lower.tail=FALSE)
> chi.tabla
[1] 23.68479
>
> # Estadistico x_0 de los pseudoaleatorios
> X_0=sum(su)
> X_0
[1] 10.682
>
```

Figura 2.12: Resultados sobre los estadísticos de prueba de χ^2 para cuadrados medios

La prueba de la χ^2 conocida como bondad de ajuste, consiste en dividir el intervalo $(0, 1)$ en **m** subintervalos, como ya antes mencionado se calcula las frecuencias de cada intervalo y se observa si se comporta como se plantea en la hipótesis y se concluye que los datos sí son modelados por una distribución uniforme sobre el intervalo $(0, 1)$.

En la Tabla 2.1 y Figura 2.12 se presentan los resultados de la prueba y se analizan si se acepta o no la hipótesis, recordemos que dicha prueba indica que si el estadístico de los números pseudoaleatorios es más pequeño que el valor estadístico de las tablas de la χ^2 que se calcula con **R**, entonces se acepta la hipótesis, de lo contrario se rechaza.

Para esta prueba el estadístico de los números pseudoaleatorios es de 10.682 y el de las tablas es de 23.68476, entonces se acepta la hipótesis, recordemos que eso es lo que se plantea como hipótesis, el estadístico que calculamos debe ser pequeño a comparación del estadístico de las tablas. Por lo tanto para esta prueba y para este generador existe suficiente evidencia de que los números generados, **SÍ** son modelados por una distribución uniforme.

Se hace un análisis para cada prueba y se ve si las tres pruebas son satisfactorias tendremos los resultados para hacer una comparación de todos los generador.

El código completo de cada prueba de aleatoriedad se presenta en el Apéndice C.

2.2.2. Productos medios

Continuamos con el segundo generador que es el de **productos medios**, este generador tiene un periodo de vida de 41350, dicho periodo se determino por ensayo y error, ya que

no existe una formula como tal, para calcularlo en los generadores no lineales, pero para los lineales si existe ciertos parámetros para para calcularlo.

La primera semilla que se utilizará es $x_0 = 38765294$ y la segunda es $x_1 = 21511340$, estas semillas deben tener ocho dígitos.

Como ya se mencionó el algoritmo fué modificado para generar números de ocho dígitos, en todos los códigos fue necesario hacer una validación, para saber la longitud de los nuevos números generados.

Ahora procedemos con la ejecución del programa en **Java**, se ingresan las dos semillas y se obtiene la simulación como se muestra en la Figura 2.13 y 2.14 y proseguimos con el análisis.

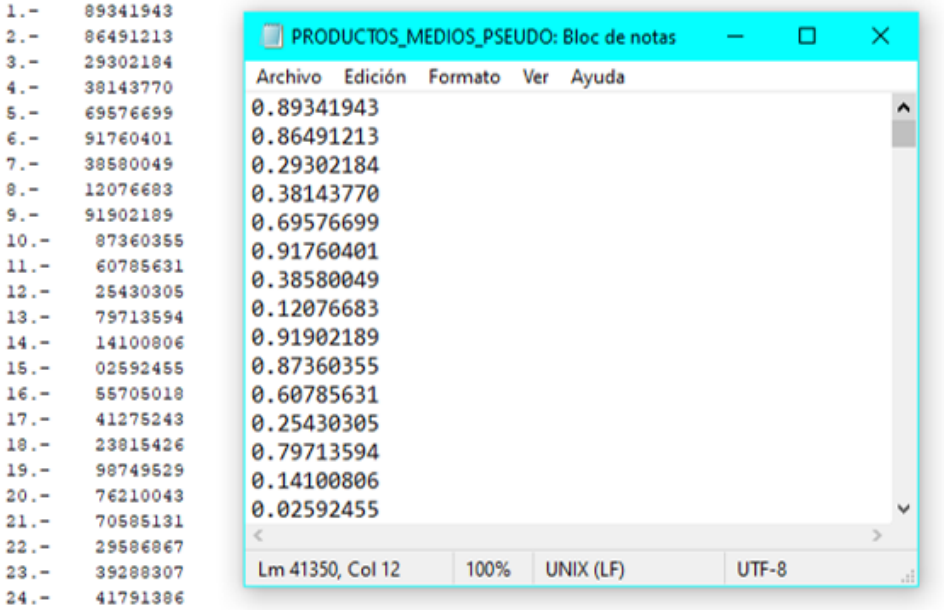


Figura 2.13: Inicio de la lista de los números generados por el algoritmo de productos medios

En la Figura 2.13 y Figura 2.14 se muestran algunos números generados y se observa que todos tienen ocho dígitos, en este caso su periodo es de 41350, que es donde termina en la Figura 2.14.

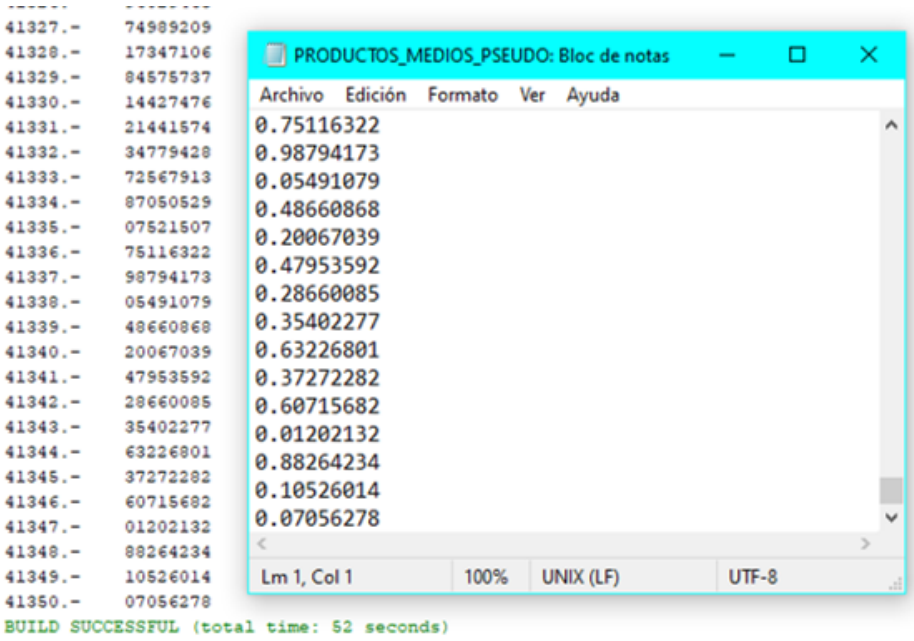


Figura 2.14: Final de la lista de los números generados por el algoritmo de productos medios

Una vez que tenemos los números pseudoaleatorios en el archivo de texto, aplicamos las pruebas de aleatoriedad.

Prueba de medias

Para la prueba de medias se puede observar en la Figura 2.15 que la media de los números pseudoaleatorios es de 0.4974971 y ésta cae dentro del intervalo de confianza, por lo tanto se acepta la hipótesis.

```
> # Intervalos
> lim.inf.pseudo<-0.5-(z/sqrt(12*n))
> lim.inf.pseudo
[1] 0.4972176
>
> lim.sup.pseudo<-0.5+(z/sqrt(12*n))
> lim.sup.pseudo
[1] 0.5027824
>
> # Media de los pseudoaleatorios
> media.pseudo<-mean(pseudoaleatorios)
> media.pseudo
[1] 0.4974971
```

Figura 2.15: Prueba de medias del algoritmo productos medios

Prueba de varianza

Para la prueba de varianza los resultados los vemos en la Figura 2.16 en ésta se muestra que la varianza de los números pseudoaleatorios también cae dentro del intervalo, por lo que no se puede rechazar la hipótesis y se acepta.

```
> #Intervalos
> lim.inf.var<-(chi.inf/(12*n-1))
> lim.inf.var
[1] 0.08219941
>
> lim.sup.var<-(chi.sup/(12*n-1))
> lim.sup.var
[1] 0.0844712
>
> #varianza de los pseudoaleatorios
> var(pseudoaleatorios)
[1] 0.08353465
```

Figura 2.16: Prueba de varianza del algoritmo productos medios

Prueba de χ^2

En la Figura 2.17 se presentan los resultados de la prueba de χ^2, dicha prueba establece que si el estadístico de los r_i generados es más chico que el estadístico de las tablas, entonces se acepta.

Este generador es diferente al anterior pues tiene 17 clases y en la Tabla 2.2 se muestra los datos agrupados por frecuencias.

```
> # Estadistico (x0^2)_(alfa,n-1)
> chi.tabla<-qchisq(alfa,k1-1,lower.tail=FALSE)
> chi.tabla
[1] 26.29623
>
> # Estadistico x_0 de los pseudoaleatorios
> X_0=sum(ecu)
> X_0
[1] 15.55526
```

Figura 2.17: Prueba de χ^2 del algoritmo productos medios

	tabla.intervalos	Oi	Ei	ecu
1	(-0.000993,0.0588]	2460	2432.353	0.31424710
2	(0.0588,0.118]	2454	2432.353	0.19265097
3	(0.118,0.176]	2515	2432.353	2.80820115
4	(0.176,0.235]	2504	2432.353	2.11042606
5	(0.235,0.294]	2406	2432.353	0.28551675
6	(0.294,0.353]	2415	2432.353	0.12379970
7	(0.353,0.412]	2464	2432.353	0.41175617
8	(0.412,0.471]	2405	2432.353	0.30759656
9	(0.471,0.529]	2415	2432.353	0.12379970
10	(0.529,0.588]	2429	2432.353	0.00462195
11	(0.588,0.647]	2409	2432.353	0.22421083
12	(0.647,0.706]	2476	2432.353	0.78321929
13	(0.706,0.765]	2358	2432.353	2.27284444
14	(0.765,0.824]	2427	2432.353	0.01178035
15	(0.824,0.882]	2337	2432.353	3.73801977
16	(0.882,0.941]	2485	2432.353	1.13951917
17	(0.941,1]	2391	2432.353	0.70305000

Tabla 2.2: Datos agrupados por frecuencias para la prueba de χ^2 para productos medios

El estadístico de los r_i asociados a los números pseudoaleatorios generados es de 15.55526 y el de las tablas es de 26.29623, entonces se acepta la hipótesis y existe suficiente evidencia de que los números generados se comportan como una distribución uniforme.

2.2.3. Multiplicador constante

Para este generador que tiene un periodo de vida de 15779, la semilla que se utilizará es $x_0 = 38765294$ y la constante será $a = 21511340$ ambos deben de ser del mismo tamaño.

Ejecutando el programa en **Java**, se obtiene lo siguiente:

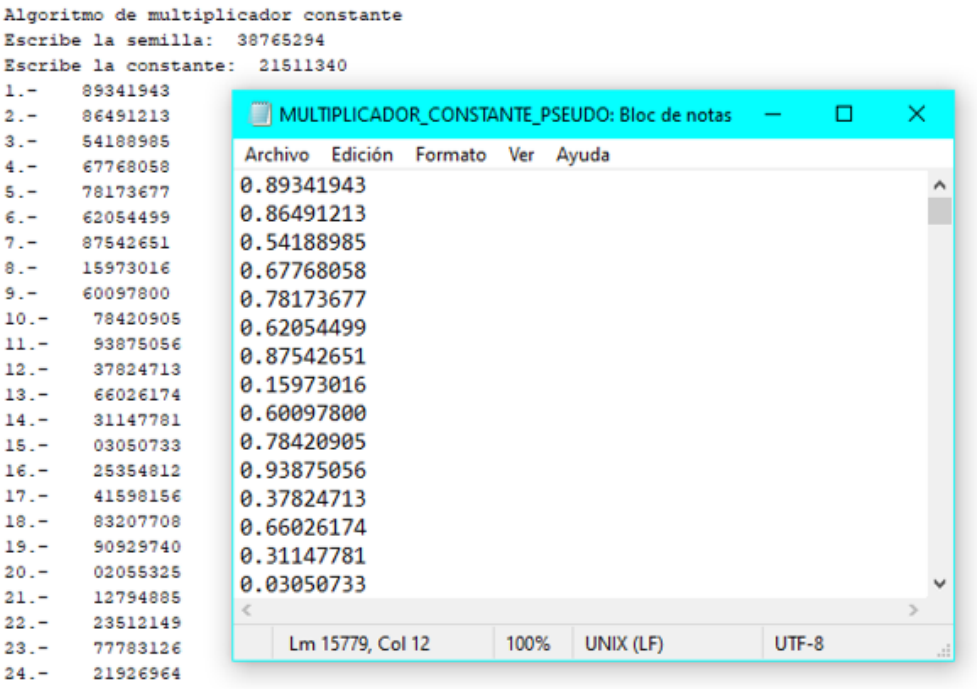


Figura 2.18: Lista de los números generados por el algoritmo de multiplicador constante

En la Figura 2.18 se muestran los números pseudoaleatorios, lo que se puede observar es que las dos primeras semillas que arroja el programa son las mismas que dio el generador pasado, eso podría hacernos pensar que generaría los mismos pero no es así, aunque esos sean iguales, los demás son diferentes entre si.

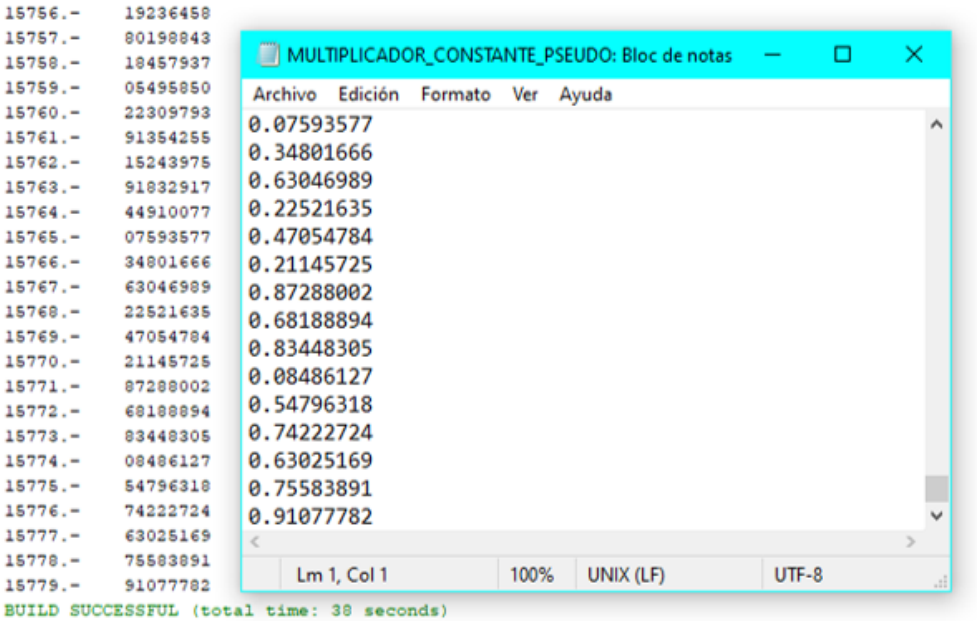


Figura 2.19: Final de la lista de los números generados por el algoritmo de multiplicador constante

Cuando tengamos los r_i aplicaremos las pruebas de aleatoriedad, esperando que en todas se acepte la hipótesis.

Prueba de media

Se puede observar en la Figura 2.20 que la media de los números generados cae dentro del intervalo, por lo que la hipótesis es aceptada y esta cerca del valor esperado.

```
> # Intervalos
> lim.inf.pseudo<-0.5-(z/sqrt(12*n))
> lim.inf.pseudo
[1] 0.4954958
>
> lim.sup.pseudo<-0.5+(z/sqrt(12*n))
> lim.sup.pseudo
[1] 0.5045042
>
> # Media de los pseudoaleatorios
> media.pseudo<-mean(pseudoaleatorios)
> media.pseudo
[1] 0.4987
```

Figura 2.20: Resultados de la prueba de medias para el algoritmo de multiplicador constante

Prueba de varianza

Para la prueba de la varianza se muestra en la Figura 2.21 que dicho número cae dentro del intervalo, por lo tanto se acepta la hipótesis la varianza es 0.08249113.

```
> #Intervalos
> lim.inf.var<-(chi.inf/(12*n-1))
> lim.inf.var
[1] 0.08149973
>
> lim.sup.var<-(chi.sup/(12*n-1))
> lim.sup.var
[1] 0.08517726
>
> #varianza de los pseudoaleatorios
> var(pseudoaleatorios)
[1] 0.08249113
```

Figura 2.21: Resultados de la prueba de varianza para el algoritmo de multiplicador constante

Prueba de χ^2

Para este generador el estadístico dio como resultado 11.50542 y el de las tablas es 23.68479, entonces se acepta la hipótesis y existe suficiente evidencia de que los números generados sí provienen de una distribución uniforme.

En la Tabla 2.3 se puede observar como fueron organizados los datos para poder sacar el estadístico y se observa en qué intervalo es dónde hay más números generados, se debe de recordar que estos número son las frecuencias observadas de cada intervalo, estas pueden variar, por la generación de los números pseudoaleatorios, ya que la frecuencia esperada era que todos tuvieran la misma cantidad de números en cada intervalo, pero no es así.

```
> # Estadistico (x0^2)_(alfa,n-1)
> chi.tabla<-qchisq(alfa,k1-1,lower.tail=FALSE)
> chi.tabla
[1] 23.68479
>
> # Estadistico x_0 de los pseudoaleatorios
> X_0=sum(ecu)
> X_0
[1] 11.50542
```

Figura 2.22: Resultados de la prueba de χ^2

	tabla.intervalos	Oi	Ei	ecu
1	(-0.000942,0.0667]	1039	1051.933	0.1590130342
2	(0.0667,0.133]	1054	1051.933	0.0040602489
3	(0.133,0.2]	1062	1051.933	0.0963347910
4	(0.2,0.267]	1046	1051.933	0.0334664216
5	(0.267,0.333]	1075	1051.933	0.5058030716
6	(0.333,0.4]	1015	1051.933	1.2967277183
7	(0.4,0.467]	1051	1051.933	0.0008281049
8	(0.467,0.533]	1101	1051.933	2.2886790460
9	(0.533,0.6]	1070	1051.933	0.3102900480
10	(0.6,0.667]	1084	1051.933	0.9775059678
11	(0.667,0.733]	1004	1051.933	2.1841730570
12	(0.733,0.8]	1088	1051.933	1.2365844899
13	(0.8,0.867]	1044	1051.933	0.0598305765
14	(0.867,0.933]	1043	1051.933	0.0758645457
15	(0.933,1]	1003	1051.933	2.2762574730

Tabla 2.3: Datos agrupados por frecuencias para la prueba de χ^2 para el multiplicador constante

2.2.4. Congruencial lineal mixto

El cuarto generador es el de **congruencial lineal mixto**, este generador es totalmente diferente a los tres generadores pasados, este generador tiene un periodo de vida de $500,000$, los parámetros que se utilizan son: $x_0 = 38765294$**,** $a = 61$**,** $c = 1$ **y** $m = 99,999,990$.

Aquí también es importante que la semilla sea de ocho dígitos y los demás parámetros deben de cumplir lo que ya se mencionó al principio de este capítulo. El código de estos programas también estará en el apéndice B, para que puedan ver en que consiste cada uno de los generadores.

Ejecutando el programa codificado en **Java** se obtiene lo siguiente:

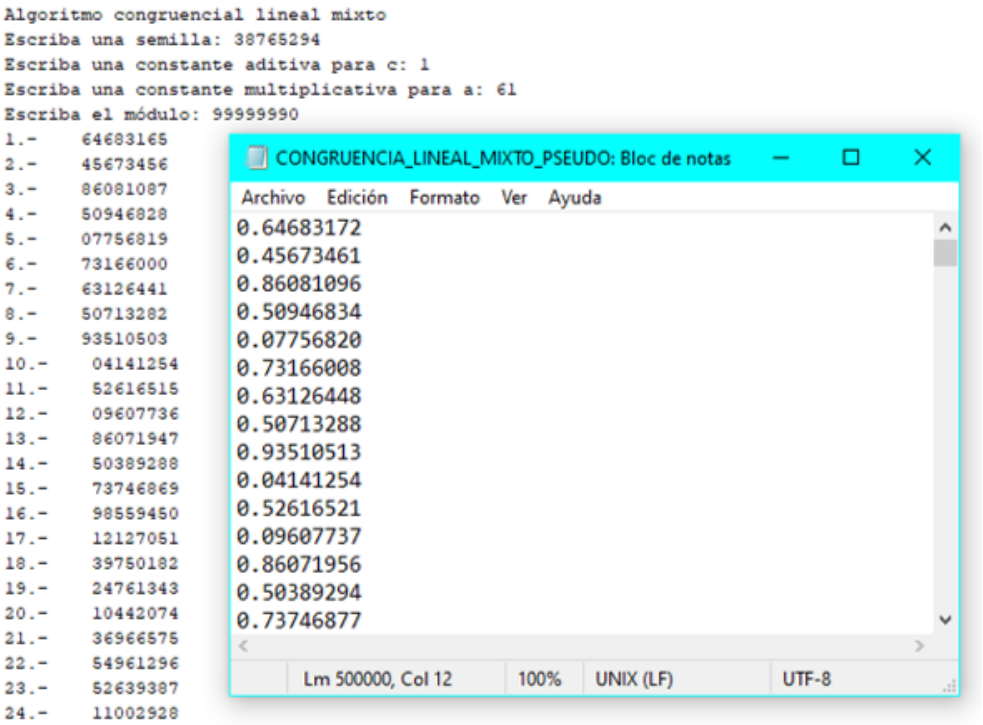


Figura 2.23: Lista de números generados por el algoritmo congruencia lineal mixto

En la Figura 2.23 se observa el despliegue de la generación de los números pseudoaleatorios, de igual forma la Figura 2.24 muestra el final de la lista. Teniendo una vez el archivo de texto con los números generados procedemos a abrirlos en R.

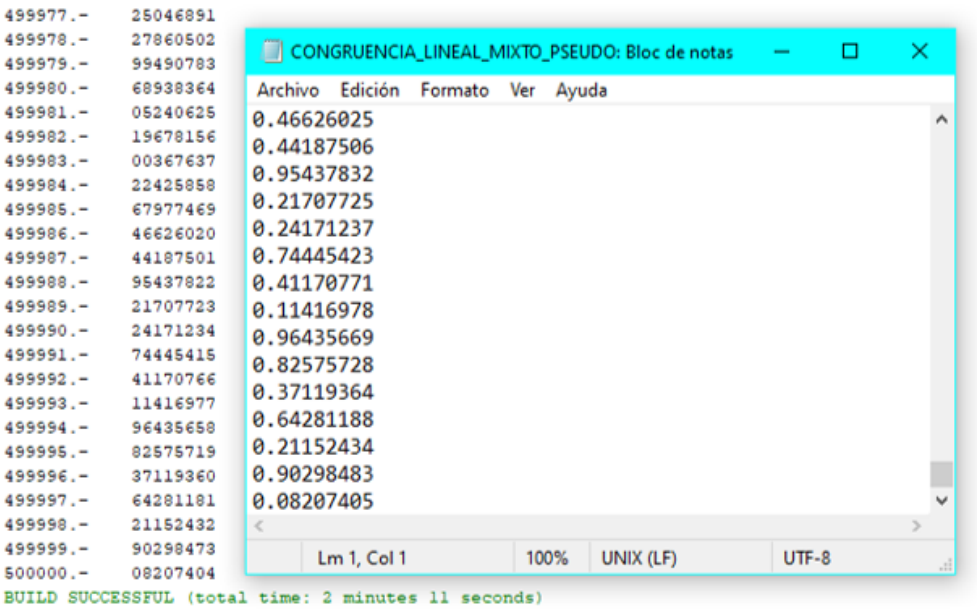


Figura 2.24: Final de los números generados por el algoritmo congruencia lineal mixto

Prueba de medias

En la Figura 2.25 se muestra que la media de los números pseudoaleatorios caen dentro del intervalo aunque está muy cercano a uno de los extremos del intervalo por décimas, aún así se acepta la hipótesis.

```
> # Intervalos
> lim.inf.pseudo<-0.5-(z/sqrt(12*n))
> lim.inf.pseudo
[1] 0.4991998
>
> lim.sup.pseudo<-0.5+(z/sqrt(12*n))
> lim.sup.pseudo
[1] 0.5008002
>
> # Media de los pseudoaleatorios
> media.pseudo<-mean(pseudoaleatorios)
> media.pseudo
[1] 0.5000584
```

Figura 2.25: Resultados de la prueba de medias para el algoritmo congruencial mixto

Prueba de varianza

En la Figura 2.26 se observa que la varianza de los números generados cae dentro del intervalo, por lo tanto se acepta la hipótesis.

```
> #Intervalos
> lim.inf.var<-(chi.inf/(12*n-1))
> lim.inf.var
[1] 0.08300684
>
> lim.sup.var<-(chi.sup/(12*n-1))
> lim.sup.var
[1] 0.08366016
>
> #varianza de los pseudoaleatorios
> var(pseudoaleatorios)
[1] 0.08334929
```

Figura 2.26: Resultados de la prueba de varianza para el algoritmo congruencial mixto

Prueba de χ^2

En la Figura 2.27 el estadístico de los números pseudoaleatorios es de 10.15888 y el de las tablas es de 30.14353, entonces se acepta la hipótesis, existe suficiente evidencia de que se comportan o modelan como una distribución uniforme.

```
> # Estadistico (x0^2)_(alfa,n-1)
> chi.tabla<-qchisq(alfa,k1-1,lower.tail=FALSE)
> chi.tabla
[1] 30.14353
>
> # Estadistico x_0 de los pseudoaleatorios
> X_0=sum(ecu)
> X_0
[1] 10.15888
```

Figura 2.27: Resultados de la prueba de χ^2 para el algoritmo congruencial mixto

	tabla.intervalos	Oi	Ei	ecu
1	(-0.000999,0.05]	25148	25000	0.87616
2	(0.05,0.1]	24893	25000	0.45796
3	(0.1,0.15]	24996	25000	0.00064
4	(0.15,0.2]	24714	25000	3.27184
5	(0.2,0.25]	25042	25000	0.07056
6	(0.25,0.3]	25031	25000	0.03844
7	(0.3,0.35]	25209	25000	1.74724
8	(0.35,0.4]	24941	25000	0.13924
9	(0.4,0.45]	25037	25000	0.05476
10	(0.45,0.5]	25060	25000	0.14400
11	(0.5,0.55]	24888	25000	0.50176
12	(0.55,0.6]	24983	25000	0.01156
13	(0.6,0.65]	25067	25000	0.17956
14	(0.65,0.7]	25105	25000	0.44100
15	(0.7,0.75]	24852	25000	0.87616
16	(0.75,0.8]	24861	25000	0.77284
17	(0.8,0.85]	25107	25000	0.45796
18	(0.85,0.9]	24992	25000	0.00256
19	(0.9,0.95]	25029	25000	0.03364
20	(0.95,1]	25045	25000	0.08100

Tabla 2.4: Tabla de frecuencias para la prueba de χ^2 para el algoritmo congruencial mixto

2.2.5. Congruencial lineal multiplicativo

El penúltimo generador es el **congruencial lineal multiplicativo**, este generador tiene un periodo de vida de $256,999$, que se obtuvo nuevamente por ensayo y error, los parámetros que se utilizaron fueron: la semilla $x_0 = 34765275$**,** $a = 123$ **y** $m = 99,999,990$.

Al ejecutar el programa en **Java**, se ingresan los parámetros indicados y se obtiene lo siguiente:

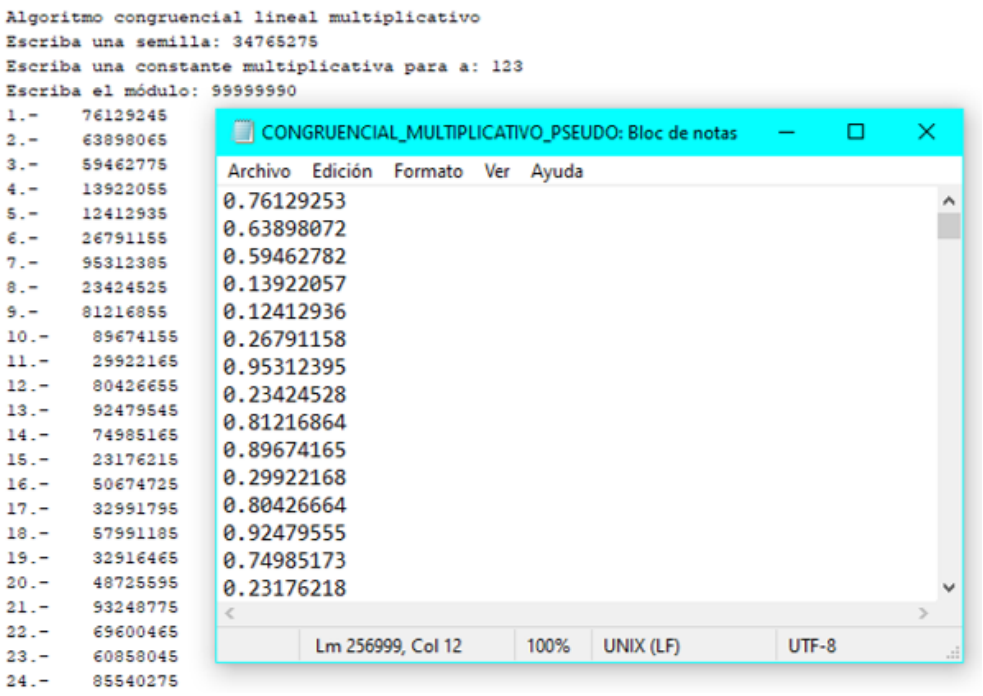


Figura 2.28: Números generados por el algoritmo congruencial lineal multiplicativo

Se observa el despliegue de la generación de los números pseudoaleatorios en la Figura 2.28, donde muestran números de ocho dígitos hasta que se cumple su periodo que se muestra en la Figura 2.29.

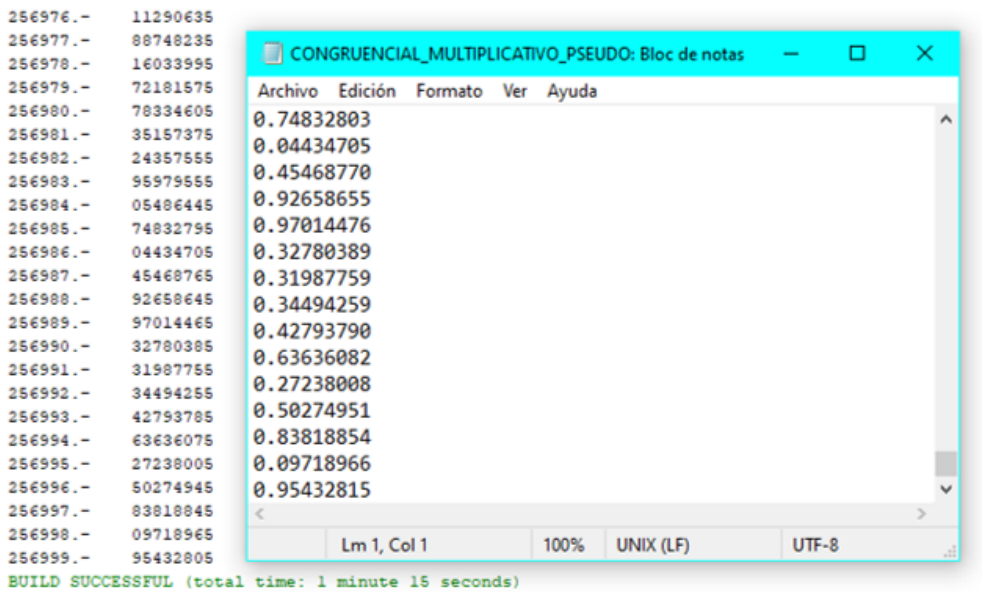


Figura 2.29: Final de la lista de los números generados por el algoritmo congruencial lineal multiplicativo

Prueba de medias

En la Figura 2.30 se muestran los resultados al ejecutar la prueba de medias en R de los números pseudoaleatorios, donde se analiza y se observa que sí cae dentro del intervalo por lo tanto se acepta la hipótesis.

```
> # Intervalos
> lim.inf.pseudo<-0.5-(z/sqrt(12*n))
> lim.inf.pseudo
[1] 0.4988839
>
> lim.sup.pseudo<-0.5+(z/sqrt(12*n))
> lim.sup.pseudo
[1] 0.5011161
>
> # Media de los pseudoaleatorios
> media.pseudo<-mean(pseudoaleatorios)
> media.pseudo
[1] 0.5009085
```

Figura 2.30: Prueba de medias para los números pseudoaleatorios generados con el algoritmo congruencial lineal multiplicativo

Prueba de varianza

Se muestra la varianza de los números generados en la Figura 2.31, se muestra el intervalo y se observa que la varianza de los números pseudoaleatorios generados está dentro del intervalo, por lo tanto se acepta la hipótesis.

```
> #Intervalos
> lim.inf.var<-(chi.inf/(12*n-1))
> lim.inf.var
[1] 0.08287802
>
> lim.sup.var<-(chi.sup/(12*n-1))
> lim.sup.var
[1] 0.08378928
>
> #varianza de los pseudoaleatorios
> var(pseudoaleatorios)
[1] 0.08336585
```

Figura 2.31: Prueba de varianza para los números pseudoaleatorios generados con el algoritmo congruencial lineal multiplicativo

Prueba de χ^2

El estadístico de los números pseudoaleatorios es de 28.76845 y el de las tablas es de 28.8693, entonces se acepta la hipótesis y existe suficiente evidencia de que los números generados sí modelan o se comportan como una distribución uniforme.

```
> # Estadistico (x0^2)_(alfa,n-1)
> chi.tabla<-qchisq(alfa,k1-1,lower.tail=FALSE)
> chi.tabla
[1] 28.8693
>
> # Estadistico x_0 de los pseudoaleatorios
> X_0=sum(ecu)
> X_0
[1] 28.76845
```

Figura 2.32: Prueba de χ^2 para los números pseudoaleatorios generados con el algoritmo congruencial lineal multiplicativo.

	tabla.intervalos	Oi	Ei	ecu
1	(-0.000991,0.0526]	13564	13526.26	0.10528179
2	(0.0526,0.105]	13483	13526.26	0.13837531
3	(0.105,0.158]	13384	13526.26	1.49625997
4	(0.158,0.211]	13426	13526.26	0.74319867
5	(0.211,0.263]	13443	13526.26	0.51254019
6	(0.263,0.316]	13534	13526.26	0.00442537
7	(0.316,0.368]	13620	13526.26	0.64959520
8	(0.368,0.421]	13468	13526.26	0.25096329
9	(0.421,0.474]	13194	13526.26	8.16181120
10	(0.474,0.526]	13495	13526.26	0.07225832
11	(0.526,0.579]	13896	13526.26	10.10665923
12	(0.579,0.632]	13623	13526.26	0.69184050
13	(0.632,0.684]	13357	13526.26	2.11810286
14	(0.684,0.737]	13466	13526.26	0.26848865
15	(0.737,0.789]	13648	13526.26	1.09563584
16	(0.789,0.842]	13619	13526.26	0.63580915
17	(0.842,0.895]	13674	13526.26	1.61361451
18	(0.895,0.947]	13549	13526.26	0.03821928
19	(0.947,1]	13556	13526.26	0.06537502

Tabla 2.5: Tabla de frecuencias para la prueba de χ^2 para el algoritmo congruencial lineal multiplicativo

2.2.6. Congruencial no lineal cuadrático

El generador **congruencial no lineal cuadrático** tiene un periodo de vida de $5,000$, este se determinó de la misma forma que los anteriores. Los parámetros son: una semilla $x_0 = 38765294$, $a = 8$, $c = 1$, $b = 494$ **y** $m = 99,999,990$.

Al ejecutar el programa en **Java**, e ingresar los parámetros se obtiene lo siguiente:

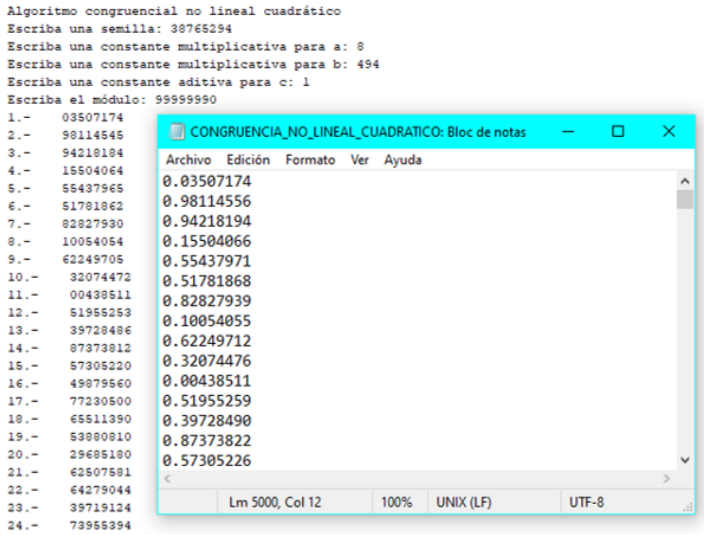


Figura 2.33: Lista de números pseudoaleatorios generados por el algoritmo congruencial no lineal cuadrático

En la Figura 2.33, se observa el despliegue de los números pseudoaleatorios, de ocho dígitos hasta que se cumple su periodo de 5000 como se observa en la Figura 2.34.

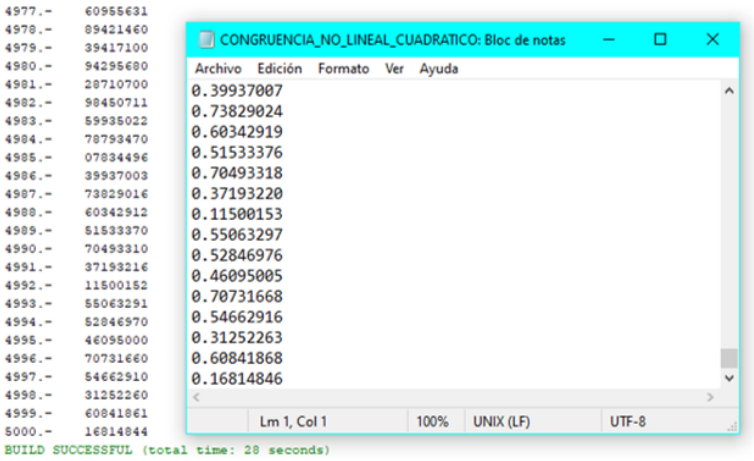


Figura 2.34: Final de la lista de los números generados por el algoritmo congruencial no lineal cuadrático

Prueba de medias

Una vez que se obtuvieron los números pseudoaleatorios se guardaron en un documento de texto para que al abrir R se apliquen las pruebas de aleatoriedad. En el caso de la prueba de medias se ve que se acepta la hipótesis por que la media de los números pseudoaleatorios cae dentro del intervalo como se observa en la Figura 2.35.

```
> # Intervalos
> lim.inf.pseudo<-0.5-(z/sqrt(12*n))
> lim.inf.pseudo
[1] 0.4919985
>
> lim.sup.pseudo<-0.5+(z/sqrt(12*n))
> lim.sup.pseudo
[1] 0.5080015
>
> # Media de los pseudoaleatorios
> media.pseudo<-mean(pseudoaleatorios)
> media.pseudo
[1] 0.5034331
```

Figura 2.35: Prueba de medias para el algoritmo congruencial no lineal cuadrático

Prueba de varianza

En la Figura 2.36 se muestra que la varianza de los números generados cae dentro del intervalo, por lo que no se puede rechazar la hipótesis.

```
> #Intervalos
> lim.inf.var<-(chi.inf/(12*n-1))
> lim.inf.var
[1] 0.08008341
>
> lim.sup.var<-(chi.sup/(12*n-1))
> lim.sup.var
[1] 0.08661585
>
> #varianza de los pseudoaleatorios
> var(pseudoaleatorios)
[1] 0.08336994
```

Figura 2.36: Prueba de varianza para los números generados por el algoritmo congruencial no lineal cuadrático

Prueba de χ^2

El estadístico de los números pseudoaleatorios es de 20.5624 y el de las tablas es de 22.36203, entonces se acepta la hipótesis y existe suficiente evidencia de que los números generados sí se modelan o se comportan como una distribución uniforme. Esto se puede apreciar en la Figura 2.37 y en la Tabla 2.6 se puede observar como se fueron creando las frecuencias para la prueba de χ^2.

```
> # Estadistico (x0^2)_(alfa,n-1)
> chi.tabla<-qchisq(alfa,k1-1,lower.tail=FALSE)
> chi.tabla
[1] 22.36203
>
> # Estadistico x_0 de los pseudoaleatorios
> X_0=sum(ecu)
> X_0
[1] 20.5624
```

Figura 2.37: Prueba de χ^2 para los números generados por el algoritmo congruencial no lineal cuadrático

	tabla.intervalos	Oi	Ei	ecu
1	(-0.00098,0.0714]	339	357.1429	0.92165714
2	(0.0714,0.143]	338	357.1429	1.02605714
3	(0.143,0.214]	368	357.1429	0.33005714
4	(0.214,0.286]	366	357.1429	0.21965714
5	(0.286,0.357]	400	357.1429	5.14285714
6	(0.357,0.429]	329	357.1429	2.21765714
7	(0.429,0.5]	324	357.1429	3.07565714
8	(0.5,0.571]	360	357.1429	0.02285714
9	(0.571,0.643]	372	357.1429	0.61805714
10	(0.643,0.714]	347	357.1429	0.28805714
11	(0.714,0.786]	378	357.1429	1.21805714
12	(0.786,0.857]	363	357.1429	0.09605714
13	(0.857,0.929]	327	357.1429	2.54405714
14	(0.929,1]	389	357.1429	2.84165714

Tabla 2.6: Frecuencias de la prueba de bondad de ajuste para el algoritmo congruencial no lineal cuadrático

SECCIÓN 2.3

¿Cuál es el mejor generador para implementar en el token?

Para tomar una decisión de cuál generador es mejor, se reunió la información de cada generador y se colocó como se muestra en la Tabla 2.7.

Generador	Periodo de vida	Tiempo	Media	Varianza	Bondad de ajuste
Cuadrados medios	10,000	36 seg	0.4975468	0.08226933	$10.682 < 23.68479$
Productos medios	41,350	52 seg	0.4974971	0.08353465	$15.55526 < 26.29623$
Multiplicador constante	15,779	38 seg	0.4987	0.08249113	$11.50542 < 23.68479$
Lineal mixto	500,000	2 min , 11 seg	0.5000584	0.08334929	$10.15888 < 30.14353$
Lineal multiplicativo	256,999	1 min , 15 seg	0.5009085	0.08336585	$28.76845 < 28.8696$
No lineal cuadrático	5,000	28 seg	0.5034331	0.08336994	$20.5624 < 22.36203$

Tabla 2.7: Información sobre los generadores

Para tomar la decisión de qué generador implementar en el token, lo que hicimos fue identificar cuál algoritmo tiene el periodo más grande, cuál es el más rápido, en los casos de la media y varianza tomamos los que estuvieran más cerca a $\frac{1}{2}$ y $\frac{1}{12}$ respectivamente, en el caso de bondad de ajuste, se identificaron los que tienen una diferencia más grande con respecto a χ^2 de las tablas.

Una vez que se tiene recopilada dicha información, se contruye la Tabla 2.8, se muestra la distancia que existe de la media y la varianza con los valores encontrados, se busca la más pequeña, de igual forma con la que diferencia que existe en bondad de ajuste, se busca la más grande.

Generador	Periodo de vida	Tiempo	Media	Varianza	Bondad de ajuste
Cuadrados medios	10,000	36 seg	0.00245320	0.001064003	13.002790
Productos medios	41,350	52 seg	0.00250290	0.000201317	10.740970
Multiplicador constante	15,779	38 seg	0.001300	0.000842203	12.179370
Lineal mixto	500,000	2 min , 11 seg	0.000058400	0.000015957	19.984650
Lineal multiplicativo	256,999	1 min , 15 seg	0.00090850	0.000032517	0.100850
No lineal cuadrático	5,000	28 seg	0.003433100	0.000036607	1.799630

Tabla 2.8: Análisis de resultados

En la Tabla 2.8 se observa que el generador lineal mixto "**gana**" en cuatro de las cinco pruebas presentadas. Por lo anterior este generador es el que se implementó en **Java** para el Token.

CAPÍTULO 3

DESARROLLO DE LA APLICACIÓN

En el Capitulo 2 se hicieron las simulaciones con todos los generadores propuestos, con sus pruebas de aleatoriedad, se determinó en qué generador se va a trabajar, éste es el algoritmo congruencial lineal mixto.

Se hizo el diseño de una aplicación de escritorio donde se implementó el generador, logrando simular el token mediante una aplicación y se observó su buen funcionamiento.

SECCIÓN 3.1

Diseño en Java de la aplicación

Para empezar se diseño la siguiente propuesta del token ver la Figura 3.1.

Partes de la aplicación:

1.- Es la **pantalla** donde se ingresan los ocho dígitos y donde se muestran los números generados

2.- Los **números** estarán funcionando cada uno con su respectivo valor

3.- El botón **CL** es para limpiar la pantalla

4.- El botón **Generar** como lo dice su nombre, genera un número pseudoaleatorio por pulsación mediante el generador definido

5.- El botón **OFF** sale de la aplicación

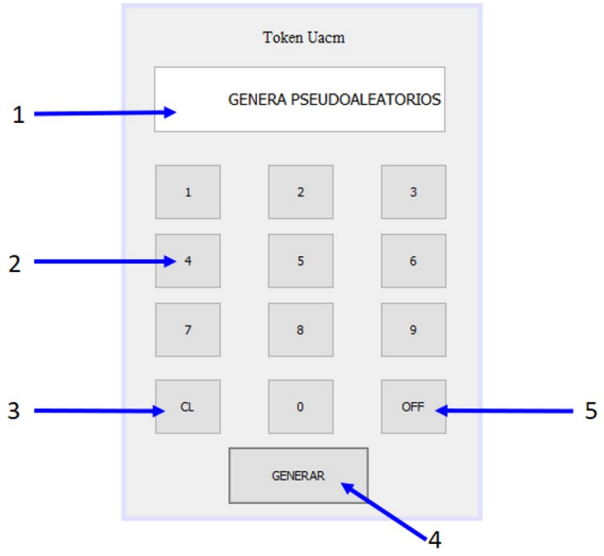


Figura 3.1: Partes del token

Una vez definido el diseño de la aplicación se hizo en Java la interfaz, ahí se programáron las funciones y métodos que harán las acciones mencionadas.

Figura 3.2: Programación de los botones del Token

Una vez programados los botones y los métodos, corresponderá hacer pruebas con el token validando que genere los ocho dígitos, esto se mostrará en pantalla.

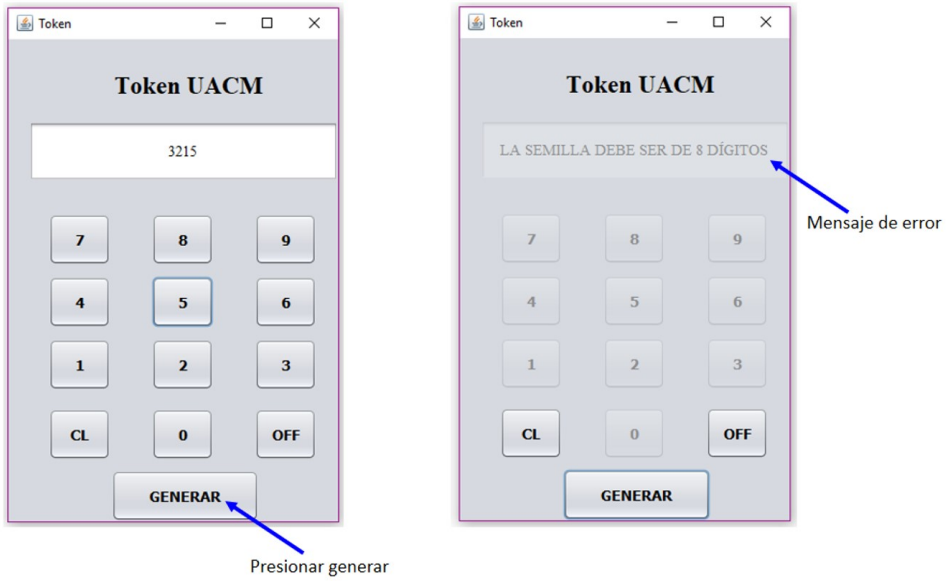


Figura 3.3: Token con una semilla de longitud menor a 8 dígitos

Al presionar el botón generar y si en pantalla no se encuentra un número de ocho dígitos, mandará un mensaje de error y solicitará que se ingrese una semilla de ocho dígitos, esto pasará con los números de mas o menos a ocho dígitos de longitud, como se muestra en las Figuras 3.3 y 3.4.

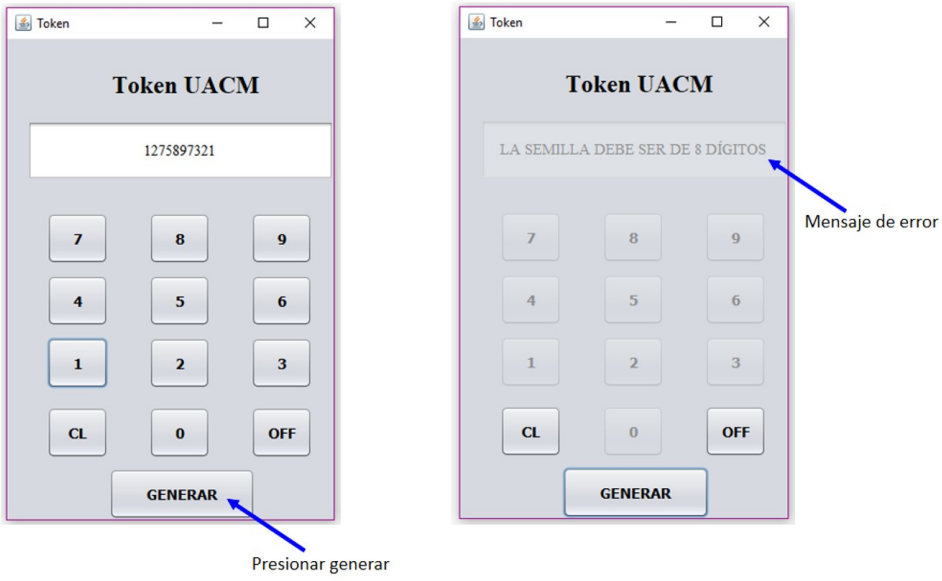


Figura 3.4: Token con una semilla de longitud mayor a 8 dígitos

Una vez que se verifica que la semilla sí es de ocho dígitos, la pantalla se inhabilitará, al igual que los botones y sólo estará disponible el botón de **generar**, **CL** y de **OFF**. El usuario decide cuando parar la generación de números pseudoaleatorios.

Nota: esta aplicación se puede personalizar con los colores que uno desee.

Se completó la programación de la aplicación de escritorio.

Figura 3.5: Token con algoritmo congruencial lineal mixto con la semilla 38765294 que da inicio a la generación de números pseudoaleatorios

SECCIÓN 3.2

Diseño de la aplicación en Android Studio

Para Android Studio al igual que en Java se trabajó sobre un proyecto y así se implementó la aplicación móvil. Se creó la aplicación, con la siguiente vista, Figura 3.6.

Sobre esta ventana se puede hacer el mismo diseño que la aplicación de escritorio, pero como es para un celular, no es necesario poner botones ya que el tiene su teclado y eso lo facilita.

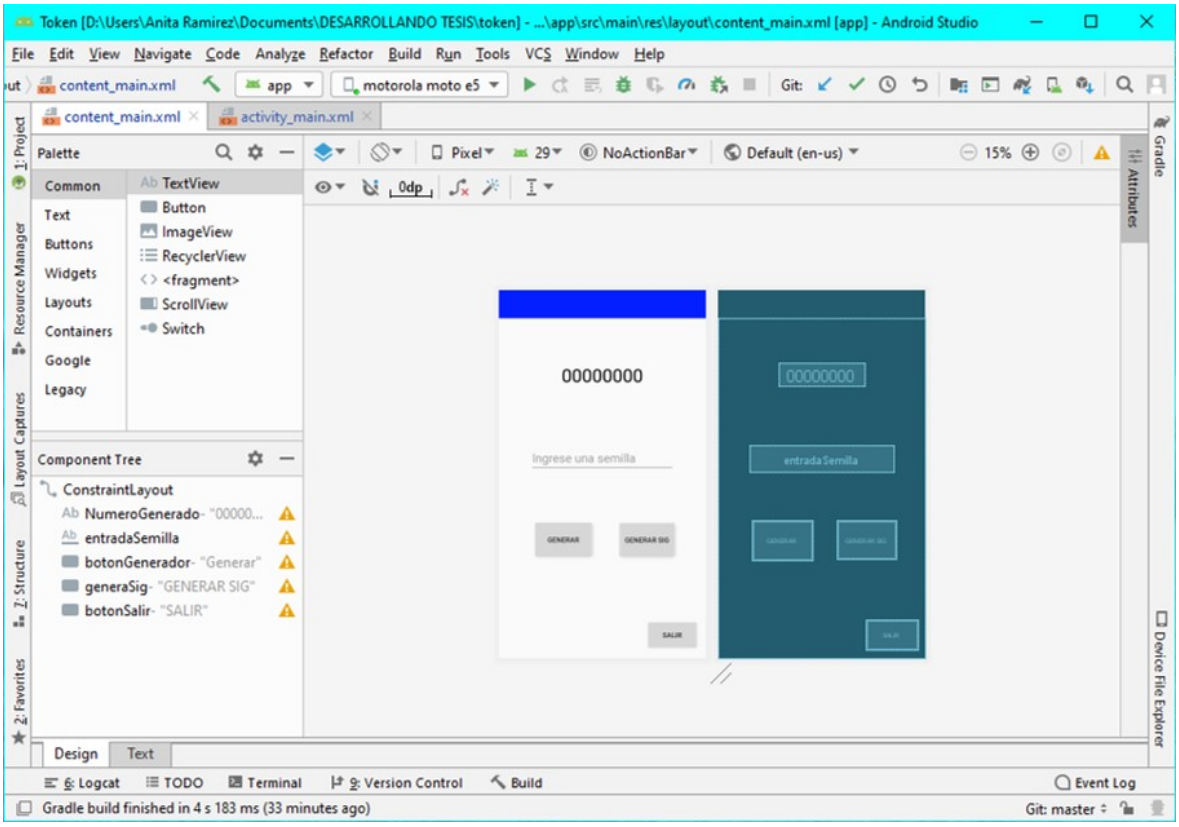


Figura 3.6: Plantilla del token para android

La implementación de Java y Android es casi la misma, sólo se hacen algunas modificaciones para la aplicación. Este puede ser ejecutado desde un emulador o desde un celular.

En la Figura 3.7 así queda la aplicación con las funciones terminadas y se ejecuta directamente desde celular.

Al ejecutarlo verificaremos que si haga lo que necesitamos, como es el caso del teclado que se puede ver en la Figura 3.8.

Figura 3.7: Token móvil android

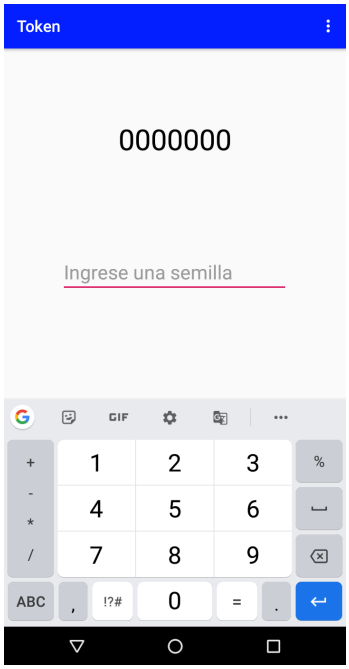


Figura 3.8: Verificación del teclado de la aplicación

Si se ingresa un número de longitud menor o mayor a ocho dígitos, lo que hace la aplicación es cerrarse, por lo que sólo acepta ocho dígitos. Continuamos ingresando la semilla y se presiona el botón generar, con lo que dará inicio a la generación de la primera semilla, observe la Figura 3.9.

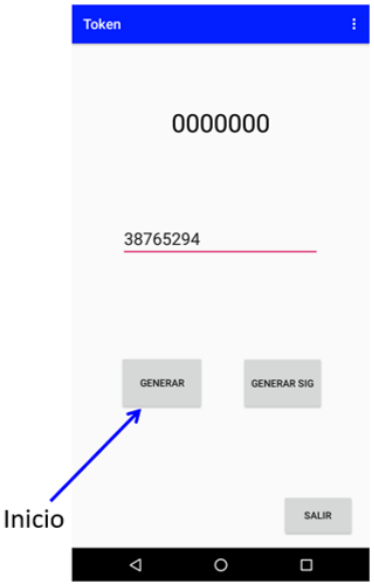


Figura 3.9: Ingresa la primera semilla

Figura 3.10: Generación de los números pseudoaleatorios

Confirmamos que sí genera el siguiente número de ocho dígitos, por lo que el otro botón se utiliza para seguir generando los números las veces que uno desee. Y por otro lado también se tiene un botón de salir de la aplicación, como se muestra en la Figura 3.10.

Se hacen todas las pruebas, para ver que funcione a la perfección y así poder usarse con la pagina web.

Lo último que se hace es personalizar la aplicación con un logo que se verá en el celular, como se muestra en la Figura 3.11.

Figura 3.11: Presentación del Token

CAPÍTULO 4

DESARROLLO DE LA PÁGINA WEB

Lo que se presenta a continuación es la implementación de una página web, la cual ayuda como complemento del token móvil. La idea principal de esta página, es que sirva como un método de acceso, en el que la página deberá generar un número pseudoaleatorio para que sea introducido a la aplicación mencionada en el Capitulo 3, el diseño que se tendrá se observa en la Figura 4.1, es un diseño básico que está compuesta de la siguiente manera:

1.- La primera pestaña es el **Inicio**, donde se hace una presentación de la página, con datos generales de la Universidad, algunas fotos del plantel, así como el nombre del proyecto, datos de la estudiante y director, que trabajaron en el proyecto entre otros datos.

2.- La segunda pestaña se llama **Historia**, donde se muestra algún acontecimiento histórico, que está relacionado con el tema principal.

3.- En la tercer pestaña se muestra una introducción donde se explica el propósito de la página y a esta se le llamará **Pública**.

4.- La última pestaña es de forma **Privada**, con ella se trabajará principalmente, aquí podemos hacer la implementación para verificar el acceso a la verdadera página privada.

Ejecución: la página da la bienvenida con un titulo, donde se encuentra un recuadro en el que dará la semilla, un número de ocho dígitos para ser introducido al token móvil, antes mencionado.

La aplicación móvil generará otro número de ocho dígitos, que será introducido en la página, si el número coincide con el número guardado que tiene la página entrará a ver que tiene de contenido, de lo contrario no le dará acceso, lo anterior se realiza verificando la llave introducida.

Nota: La página mostrará un mensaje donde dirá si coincide o no el número introducido con la lleve de la página.

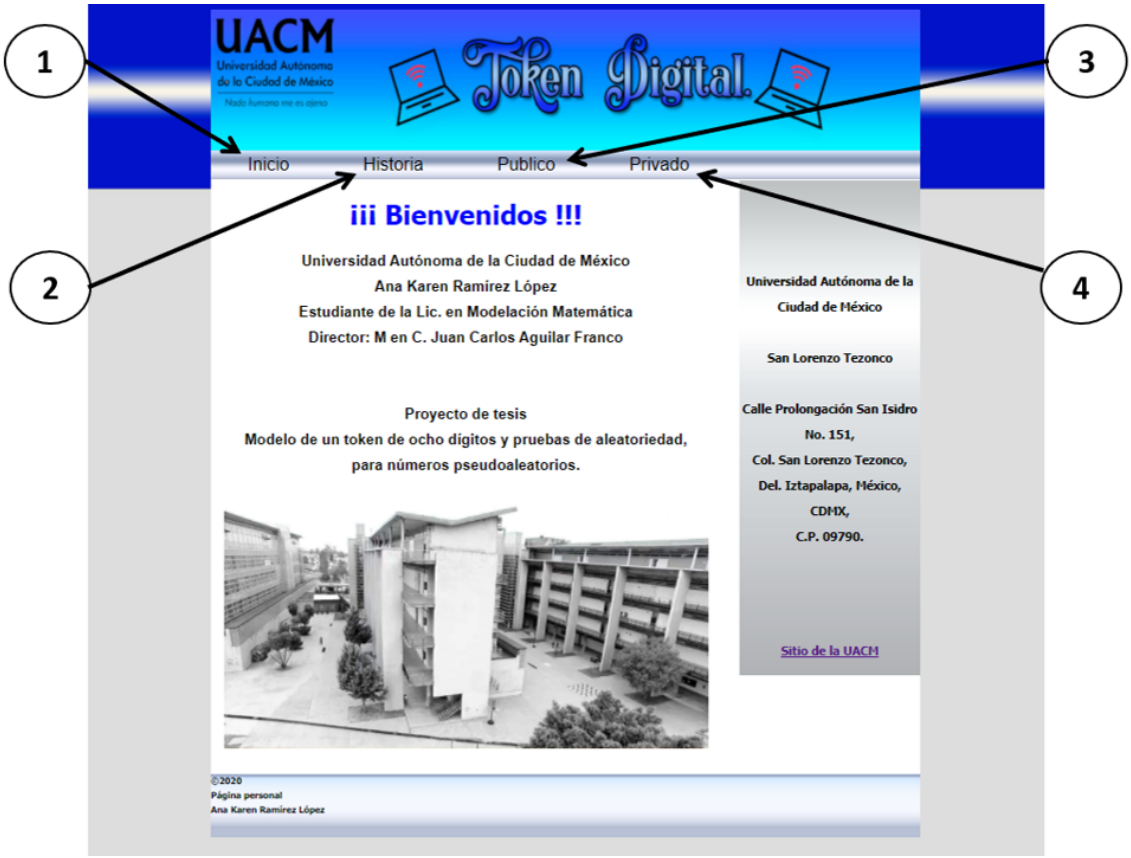


Figura 4.1: Página web

Todo el diseño se realizo en Adobe Photoshop, donde se empieza a planear el diseño, las secciones, el contenido de la página, entre otras cosas. Para la conexión de la página, se necesitan tres programas: **Apache Maven, Wampserver y Spring-tool**. A continuación se mostrará un poco de lo que es cada programa.

SECCIÓN 4.1

Apache Maven Proyect

4.1.1. ¿Qué es Apache Maven?

"**Apache Maven es una herramienta que estandariza la configuración de un proyecto en todo su ciclo de vida**, como por ejemplo en todas las fases de compilación y empaquetado, la instalación de mecanismos de distribución de librerías para que puedan ser utilizadas por otros desarrolladores y equipos de desarrollo." [7]

En Resumen es una herramienta que ayuda a la construcción de proyectos java, creada por Jason Van Zyl en el 2002.

4.1.2. Características de Maven

Maven es la base de los compiladores, pues éste ofrece un soporte gracias a algunas características:

- Es multi-plataforma (linux, windows)
- Es compatible con multiples IDES (como eclipse, IntelliJ y NetBeans)
- Se puede reutilizar código y se pueden ocupar librerías
- Apache Maven ofrece repositorios oficiales y públicos
- Es un software libre.

SECCIÓN 4.2

Wampserver

4.2.1. ¿Qué es Wampserver?

"WampServer es un entorno de desarrollo web para Windows en el cual se podrán crear aplicaciones web con Apache, PHP y base de datos en MySQL (motor de base de datos)." [8]

WAMP funge como un servidor virtual local, antes de ser hospedado a un hosting y así se pueda visualizar y trabajar de manera segura.

Es de fácil instalación sólo hay que seguir las indicaciones, a la hora de ejecutarlo.[1]

4.2.2. Características de Wamp

- Manejo de bases de datos con MySQL
- Software para servidor web Apache
- Es software libre

SECCIÓN 4.3

Spring Tool Suite

4.3.1. ¿Qué es Spring Tool Suite?

Spring es un framework para el desarrollo de aplicaciones, de código abierto, pueden ser usadas en cualquier aplicación desarrollada en Java.

"El framework Spring ofrece soluciones bien documentadas y prácticas comunes utilizadas en la industria, por lo que se ha considerado incluso como una alternativa a Enterprise JavaBeans." [9]

Para instalarlo se deben de seguir las instrucciones que salen al ejecutar el archivo de instalación.[2]

4.3.2. Características de Spring Tool Suite

- Es compatible con java o cualquier otro IDE (NetBeans, Eclipse)
- Es soporte de aplicación web, en un lenguaje HTML
- Permite acceder a base de datos

[1] Pueden revisar la bibliográfia **[9]** para seguir los pasos.
[2] También pueden revisar la bibliográfia **[10]** para seguir los pasos.

- Es multi-plataforma (Linux, Mac OSX o Windows XP/Vista/7)

- Es software libre

Dichos software cuentan con más funciones, pero esto es suficiente para obtener un modelo, la Vista y el Controlador, para la página, que es en lo que ayuda Spring.

SECCIÓN 4.4

Implementación en Spring

Una vez instalado los tres programas comenzamos a trabajar con Spring, se debe crear un nuevo proyecto, en él se hará todo el montaje de la página, la primera vista que se tiene al abrir el spring es como se muestra en la Figura 4.2.

Primero crearemos un proyecto Maven como se muestra en la Figura 4.3, donde se le dará un nombre al proyecto, y algunas especificaciones que pide el programa entre las que se encuentra cómo y dónde se guardará para finalmente crear el proyecto.

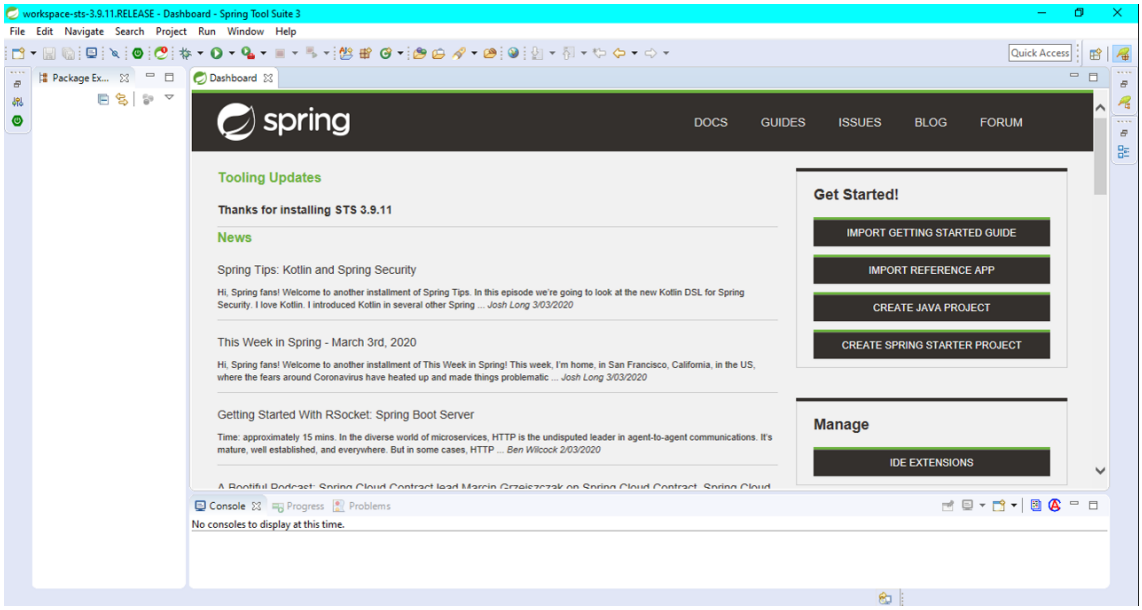


Figura 4.2: Vista Spring

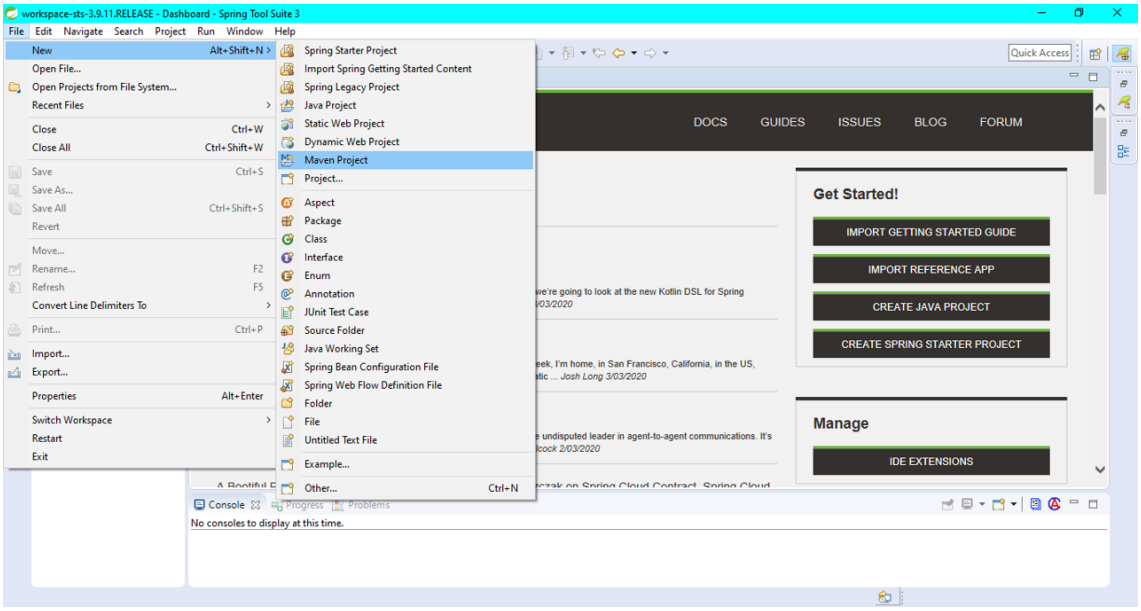


Figura 4.3: Creación de Proyecto Maven

Como se muestra en la Figura 4.4 el Proyecto se llama “Proyecto Token”, sobre él creamos un nuevo archivo, aquí damos inicio a la codificación de la página con el lenguaje que se usará que es **HTML** y la primera pestaña que se creará es la de home.html.

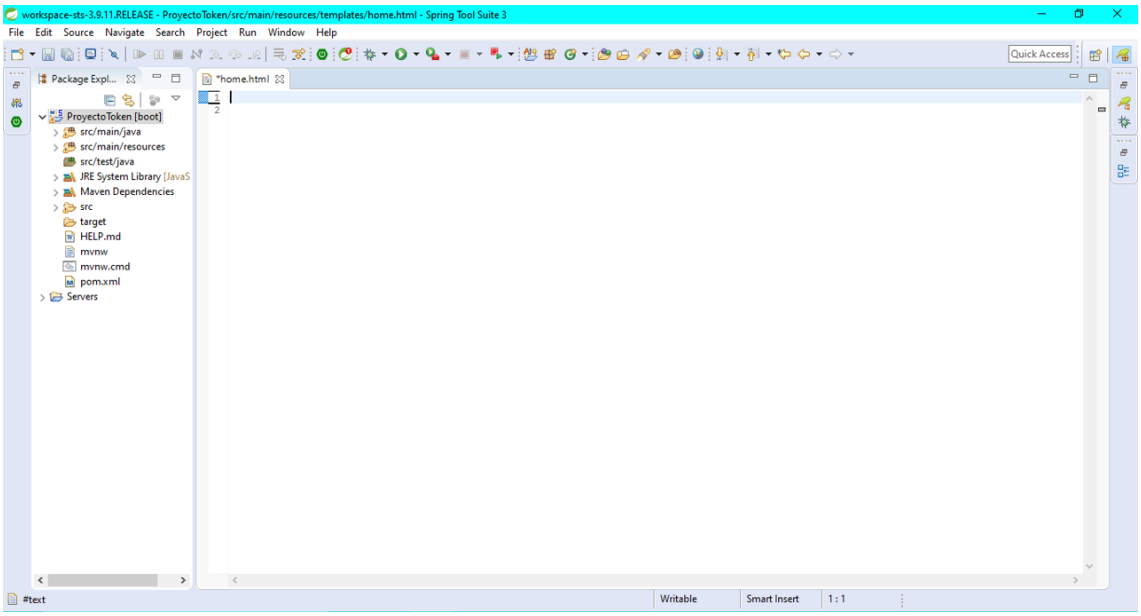


Figura 4.4: Pestaña home.html

Si recordamos al inicio del capítulo se mencionó que eran cuatro pestañas las que se iban a crear, los archivos se llamarán; home.html, historia.html, publico.html, privado.html y

pagprivado.html, esta última no se podrá ver a menos que se permita el acceso desde la página privado.html. Todos ellos tendrán el mismo diseño y se harán los enlaces entre si. Como se muestra en la Figura 4.5.

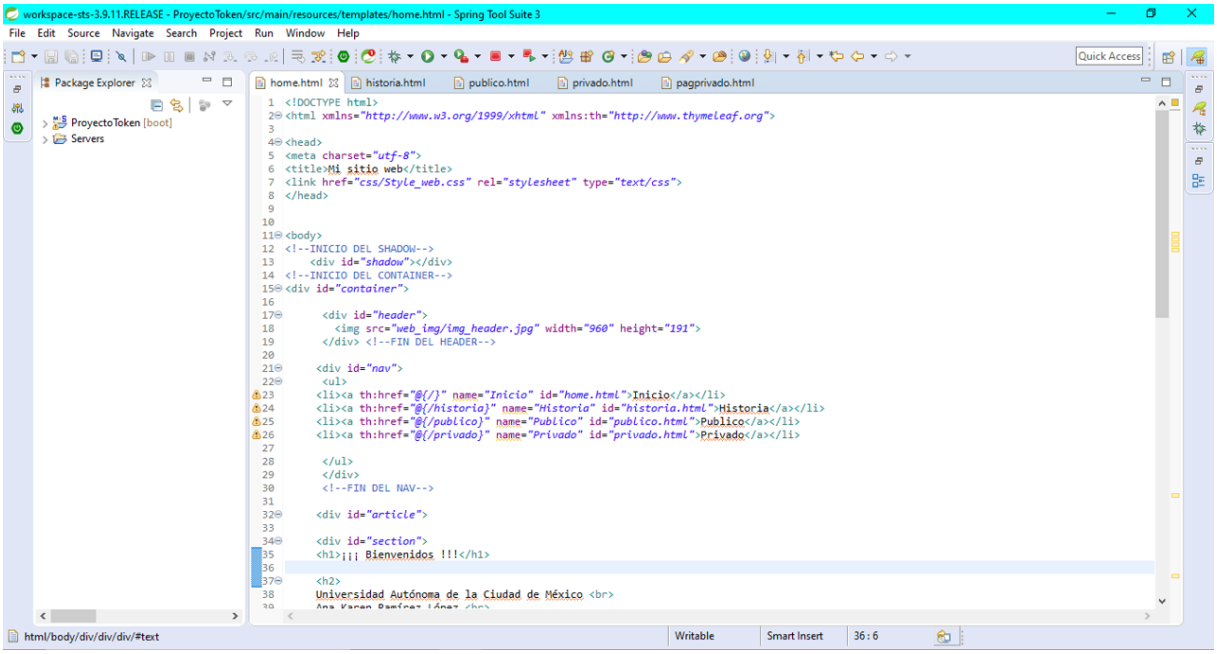


Figura 4.5: Archivos .html

Aunque todas las pestañas tengan el mismo diseño, no todas llevan el mismo contenido. La vista de la primera pestaña es la Figura 4.1.

La segunda vista es la Figura 4.6, es la página que muestra una breve introducción, donde explica los primeros antecedentes de los números aleatorios. Recordemos que no son los mismos, estos dan pie para abordar el tema de este trabajo, que ya fue desarrollado en el Capítulo 1 y Capítulo 2.

En la siguiente página, que es la pública, se encontrará una breve introducción del por qué se realizó la elaboración de la página; lo que respalda el trabajo realizado, además de hacer una implementación el token, recordando que son necesarias las página web y la aplicación móvil, pero la vista de esta página es la de la Figura 4.7.

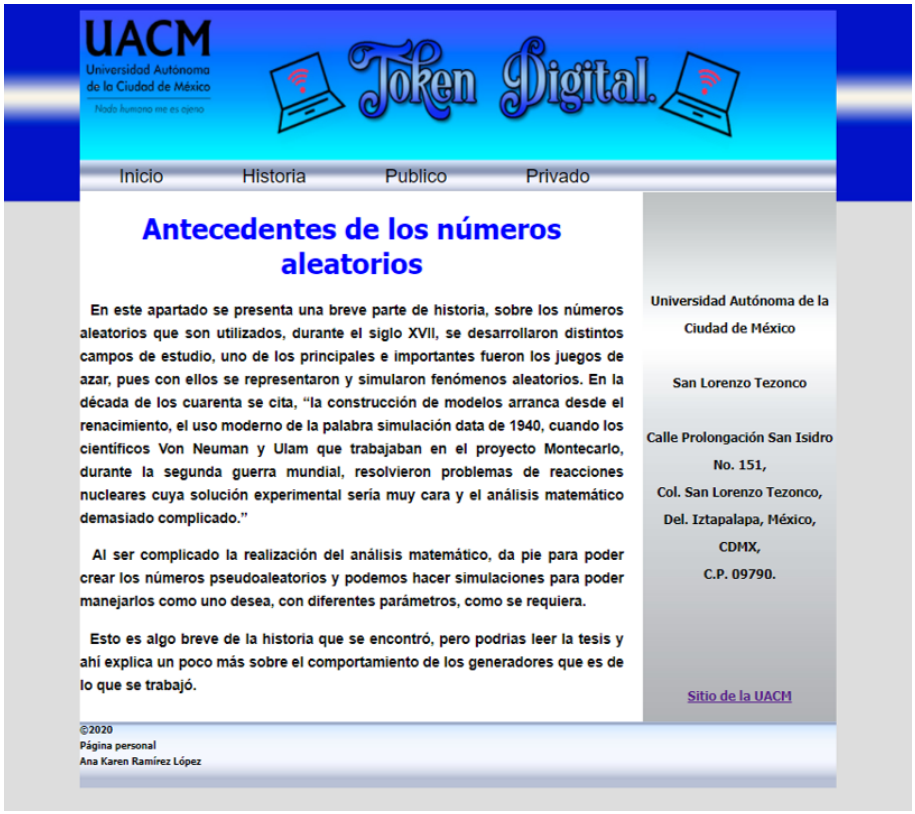


Figura 4.6: Historia

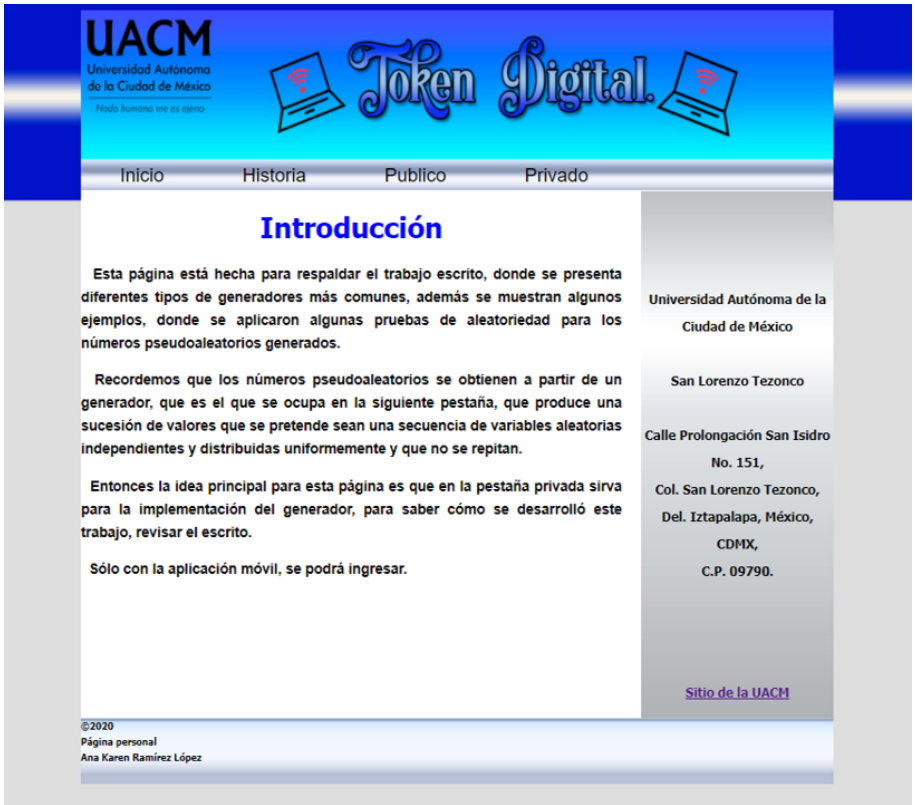


Figura 4.7: Pública

En la última página que es la privada, contará con dos cajas o contenedores, cada una de ellas, tendrá una caja de texto y un botón respectivamente, sus funciones serán sencillas, en una de ellas dará la semilla que se deberá ingresar al celular, después el nuevo número que dé la aplicación se introducirá como la nueva semilla generada.

Esta semilla funge como una llave, para que nos enlace con la página privada, como se muestra en la Figura 4.12, así que nuestra ventana llave será la Figura 4.8.

Hasta este punto ya tenemos las cuatro páginas armadas y enlazadas, ahora lo que sigue es hacer pruebas para detectar posibles errores y validar el acceso de forma correcta.

Figura 4.8: Página llave

Una vez que tenemos montada la página y que verificamos que si corre el código del generador, ahora hacemos pruebas con diferentes semillas para poder verificar que da acceso de forma correcta. En la Figura 4.9 se muestra la semilla de un número de ocho dígitos, el número es: **07162563**, ¿ahora qué pasa?.

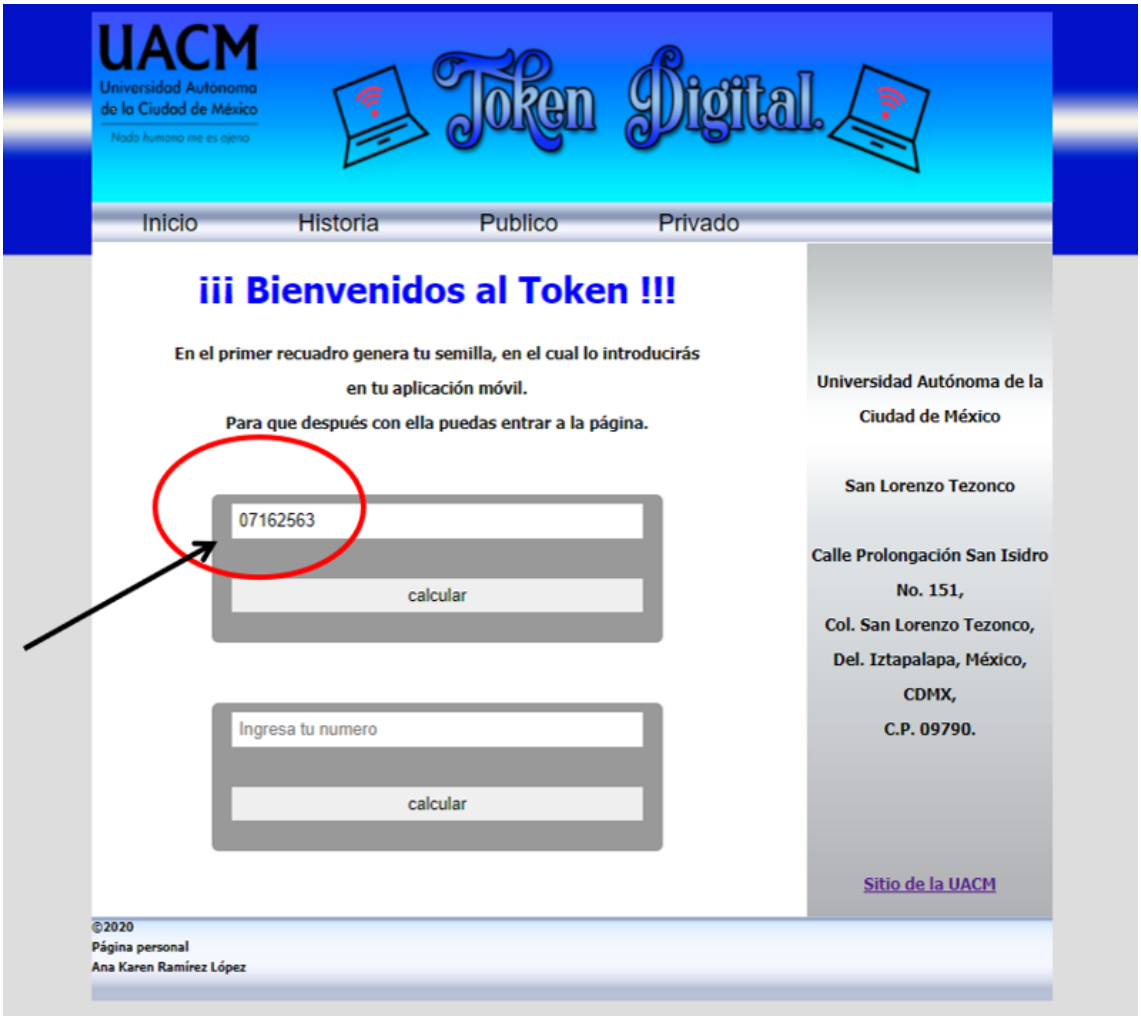


Figura 4.9: Primera simulación

Uno puede ingresar cualquier número de ocho dígitos, y no podrá entrar a la página, por que no hay forma de adivinar el número, ya que es necesario la aplicación y nos marcará un mensaje donde diga que es **erróneo el número que ingresaste**, como se muestra en la Figura 4.10. Y así podremos intentar con diferentes semillas, no dará acceso.

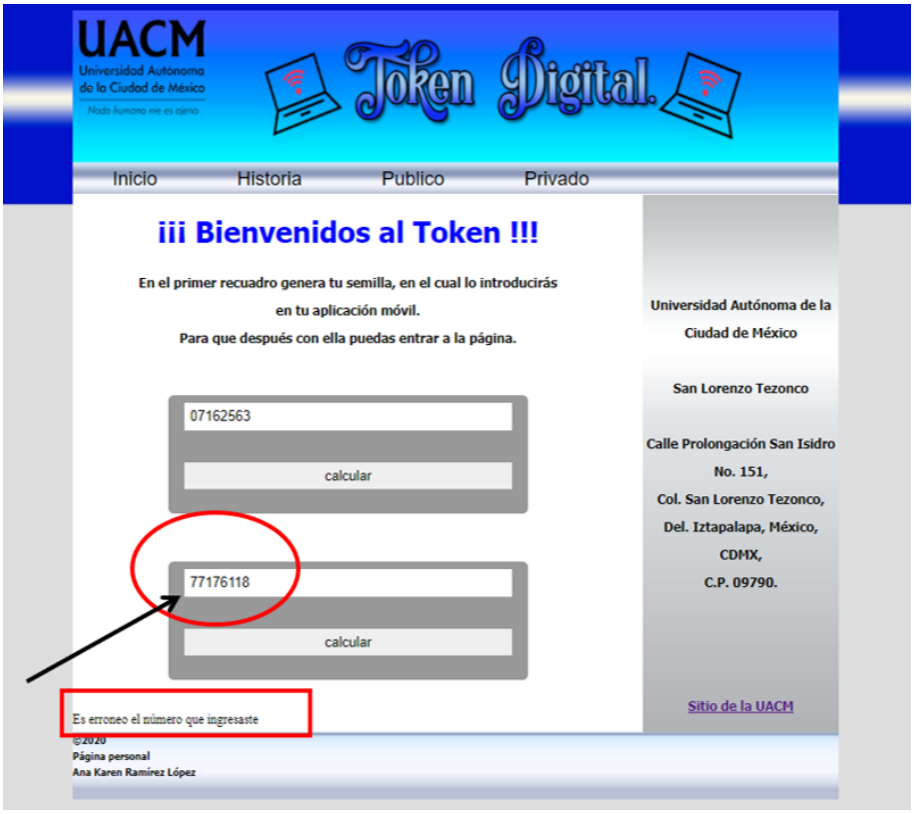


Figura 4.10: Error en la semilla

Recordemos que en el Capitulo 2 hicimos varias pruebas, donde ocupamos parámetros necesarios para el generador que cumplió todas las pruebas de aleatoriedad. Y quien no conozca el mecanismo del generador no podrá saber cuál es la semilla nueva.

Ahora hagamos el caso de que contamos con la aplicación, se ingresará el número **07162563**, presionamos el botón de generar y el nuevo número que nos da es **36916384** que es el que se muestra en la Figura 4.11.

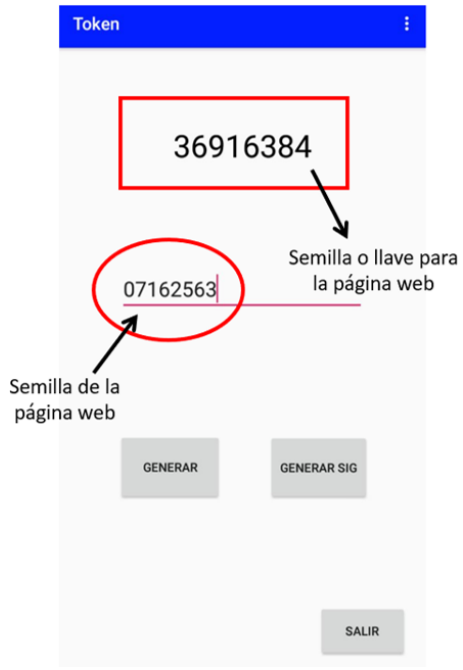


Figura 4.11: Nueva semilla

En la Figura 4.12 se podrá observar un mensaje de felicidades, donde dice que acaba de entrar a la página privada con el número que generó la aplicación, dado que lo que hace el código del generador es verificar que los números coincidan, para este caso lo único que contiene la página es el código del generador que se utilizó y que viene en la tesis.

Figura 4.12: Página privada

CONCLUSIÓN

En esta tesis se presentó lo importante que son los números pseudoaleatorios que a pesar de creer que son fáciles de encontrar o generar llevan su tiempo y su trabajo, para que estos funcionen como uno desea, ya que se pueden presentar problemas a lo largo de la investigación.

Al iniciar esta investigación, la idea principal fue muy clara y fue saber cómo generar números pseudoaleatorios, de una manera fácil, rápida y confiable, en el transcurso se presentaron diferentes problemáticas, la recolección de información, sobre cuántos generadores existían, sobre cuántas pruebas de aleatoriedad funcionaban, entre otras.

También se presentaron ventajas y desventajas, al inicio de la implementación de los generadores, no fue sencilla ya que nos enfrentamos a problemas informáticos, que debían solucionarse, ya sea por falta de memoria, por caracteres que no iban, cosas que de un principio se pensó que pasarían.

De alguna manera se cree que es fácil la manipulación de estos números, pero hay qué saber lo que se está trabajando. Aquí influyeron las pruebas de estadística que tienen un peso importante, ya que al no entender qué se estaba validando o qué se está probando, se pudo haber dicho que todas funcionaban, mientras la realidad fuera otra. Recordemos que siempre lo que un usuario o cliente busca son métodos eficientes, rápidos, concisos, eficaces y seguros.

Esta investigación se puede continuar trabajando e implementar en el área bancaria o criptográfica, sobre la parte de seguridad, el cómo proteger los números generados o cómo proteger los generados trabajados, sin que rompa la seguridad, un ejemplo puede ser el mismo token bancario, que debe terner una seguridad buena, para poder realizar varias transacciones, así este tema puede continuarlo cualquier compañero que tenga conocimientos de matemáticas, programación y estadística, ya que son los temas fundamentales que se tocan.

Mi perspectiva de esta investigación fue creer que iba a ser una forma sencilla de trabajar con el tema, más no había pensado hasta que grado esto iba a llegar, ya que ocupe todo lo que aprendí durante la carrera y hasta más. Lo que no sabía tuve que aprenderlo durante la investigación, para después implementarlo.

Mi logro principal en esta investigación es el poder aplicar todo lo aprendido y poder ver el resultado, que sí fue como lo imaginamos y estoy satisfecha con el resultado que realice en compañia de mi director y lectores.

Apéndices

APÉNDICE A

CÓDIGOS DE LOS GENERADORES DE 4 DÍGITOS

Los códigos de los algoritmos se encuentran codificados en **"C"** uno de los aportes que se realiza es su codificación en **"Java"**, todas las codificaciones son elaboración propia.

SECCIÓN A.1

Cuadrados medios

Este código es para la generación de números pseudoaleatorios de cuatro dígitos por medio del algoritmo de cuadrados medios.

```
1  package cuadradosmedios;
2
3  import java.io.PrintWriter;
4  import java.io.FileWriter;
5  import java.util.*;
6  import java.text.*;
7  import java.io.*;
8
9  /**
10  * @author Anita Ramirez
11  */
```

```
12 public class Cuadradosmedios {
13
14     //En la cadena busca un caracter E
15     public int busca(String cadena) {
16         int n;
17         n = cadena.indexOf("E");
18         return (n);
19     }
20
21     public static void main(String[] args) {
22         Scanner entrada = new Scanner(System.in);
23         Locale.setDefault(Locale.US);
24         DecimalFormat num = new DecimalFormat("####.0");
25
26         //Declaraci\'on de variables
27         String semilla, cuadsemilla, resultado;
28         int tam2semilla, tam1semilla, i, primerc, k;
29         double numero1semilla, numero2semilla;
30         double Nr = 0;
31
32         Cuadradosmedios a = new Cuadradosmedios();
33
34         //Lo que hace esta parte de c\'odigo, es que guarda los n\'umeros en
     un block de notas
35         FileWriter fichero = null;
36         PrintWriter pw = null;
37         try {
38             fichero = new FileWriter("CUADRADOS_MEDIOS_PSEUDO.txt");
39             pw = new PrintWriter(fichero);
40             System.out.println("Algoritmo de cuadrados medios");
41             System.out.printf("Escriba semilla:  ");
42             semilla = entrada.next(); //Guarda la semilla en la cadena
43             tam1semilla = semilla.length(); //Longitud de semilla
44             numero1semilla = Integer.parseInt(semilla); //Convierte la
    cadena a numero
45
46             //M\'etodo de cuadrados medios
47             for (i = 1; i <= 30; i++) {
```

```
48                numero2semilla = Math.pow(numero1semilla, 2);
49                cuadsemilla = Double.toString(numero2semilla);
50                k = a.busca(cuadsemilla);
51
52                if (k != -1) {
53                    cuadsemilla = num.format(numero2semilla);
54                }
55                tam2semilla = cuadsemilla.length() - 2;
56
57                if (tam2semilla < 8) {
58                    primerc = (tam2semilla - tam1semilla) / 2;
59                    resultado = cuadsemilla.substring(primerc, primerc +
    tam1semilla);
60                    System.out.println(i + ".-     " + resultado);
61                    numero1semilla = Double.parseDouble(resultado);
62                    Nr = (numero1semilla / 10000.0);
63                    pw.printf("%.4f \n", Nr);
64                    numero2semilla = numero1semilla;
65                } else {
66                    primerc = (tam2semilla - tam1semilla) / 2;
67                    resultado = cuadsemilla.substring(primerc, primerc +
    tam1semilla);
68                    System.out.println(i + ".-     " + resultado);
69                    numero1semilla = Double.parseDouble(resultado);
70                    Nr = (numero1semilla / 10000.0);
71                    pw.printf("%.4f \n", Nr);
72                    numero2semilla = numero1semilla;
73                }
74            }
75        } catch (Exception e) {
76            e.printStackTrace();
77        } finally {
78            try {
79                if (null != fichero) {
80                    fichero.close();
81                }
82            } catch (Exception e2) {
83                e2.printStackTrace();
```

```
84              }
85          }
86      }
87  }
```

SECCIÓN A.2

Productos medios

```
1  package productosmedios;
2
3  import java.io.FileWriter;
4  import java.io.PrintWriter;
5  import java.util.*;
6  import java.text.*;
7  import java.io.*;
8
9  /**
10  * @author Anita Ramirez
11  */
12 public class Productosmedios {
13
14     //En la cadena busca un caracter E
15     public int busca(String cadena) {
16         int n;
17         n = cadena.indexOf("E");
18         return (n);
19     }
20
21     public static void main(String[] args) {
22         Scanner entrada = new Scanner(System.in);
23         Locale.setDefault(Locale.US);
24         DecimalFormat num = new DecimalFormat("####.0");
25
26         //Declaraci\'on de variables
27         String semilla1, semilla2, prodsemilla, resultado;
28         int tam1semilla1, tam2semilla2, tamprodsemilla, i, primerc, k;
29         double Nsemilla1, Nsemilla2, Nsemilla3, PROsemilla;
```

```
30          double Nr = 0;
31
32          Productosmedios a = new Productosmedios();
33
34          //Lo que hace esta parte de c\'digo, es que guarda los n\'umeros en
    un block de notas
35          FileWriter fichero = null;
36          PrintWriter pw = null;
37          try {
38              fichero = new FileWriter("PRODUCTOS_MEDIOS_PSEUDO.txt");
39              pw = new PrintWriter(fichero);
40
41              System.out.println("Algoritmo de productos medios");
42              System.out.printf("Escribe la semilla1:  ");
43              semilla1 = entrada.next(); //Guarda la semilla1 en la cadena1
44              tam1semilla1 = semilla1.length(); //Longitud de semilla1
45              Nsemilla1 = Integer.parseInt(semilla1); //Convierte la cadena1 a
     numero
46
47              System.out.printf("Escribe la semilla2:  ");
48              semilla2 = entrada.next(); //Guarda la semilla2 en la cadena2
49              tam2semilla2 = semilla2.length(); //Longitud de semilla2
50              Nsemilla2 = Integer.parseInt(semilla2); //Convierte la cadena1 a
     numero
51
52              //M\'etodo de productos medios
53              for (i = 1; i <= 30; i++) {
54                  PROsemilla = (Nsemilla1 * Nsemilla2);
55                  prodsemilla = Double.toString(PROsemilla);
56                  k = a.busca(prodsemilla);
57
58                  if (k != -1) {
59                      prodsemilla = num.format(PROsemilla);
60                  }
61                  tamprodsemilla = prodsemilla.length() - 2;
62
63                  if (tamprodsemilla < 8) {
64                      primerc = (tamprodsemilla - tam1semilla1) / 2;
```

```
65                      resultado = prodsemilla.substring(primerc, primerc +
    tam1semilla1);
66                      System.out.println(i + ".-     " + resultado);
67                      Nsemilla3 = Double.parseDouble(resultado);
68                      Nr = (Nsemilla3 / 10000.0);
69                      pw.printf("%.4f \n", Nr);
70                      Nsemilla1 = Nsemilla2;
71                      Nsemilla2 = Nsemilla3;
72                  } else {
73                      primerc = (tamprodsemilla - tam1semilla1) / 2;
74                      resultado = prodsemilla.substring(primerc, primerc +
    tam1semilla1);
75                      System.out.println(i + ".-     " + resultado);
76                      Nsemilla3 = Double.parseDouble(resultado);
77                      Nr = (Nsemilla3 / 10000.0);
78                      pw.printf("%.4f \n", Nr);
79                      Nsemilla1 = Nsemilla2;
80                      Nsemilla2 = Nsemilla3;
81                  }
82              }
83          } catch (Exception e) {
84              e.printStackTrace();
85          } finally {
86              try {
87                  if (null != fichero) {
88                      fichero.close();
89                  }
90              } catch (Exception e2) {
91                  e2.printStackTrace();
92              }
93          }
94      }
95  }
```

SECCIÓN A.3

Multiplicador constante

```
1  package multiplicadorconstante;
2
3  import java.io.FileWriter;
4  import java.io.PrintWriter;
5  import java.util.*;
6  import java.text.*;
7  import java.io.*;
8
9  /**
10  * @author Anita Ramirez
11  */
12 public class Multiplicadorconstante {
13
14     //En la cadena busca un car\'acter E
15     public int busca(String cadena) {
16         int n;
17         n = cadena.indexOf("E");
18         return (n);
19     }
20
21     public static void main(String[] args) {
22         Scanner entrada = new Scanner(System.in);
23         Locale.setDefault(Locale.US);
24         DecimalFormat num = new DecimalFormat("####.0");
25
26         //Declaracion de variables
27         String semilla1, constante, multisemilla, resultado;
28         int tam1semilla1, tam2constante, tammultisemilla, i, primerc, k;
29         double Nsemilla1, Nconstante, Nsemilla3, MULTIsemilla;
30         double Nr = 0;
31
32         Multiplicadorconstante a = new Multiplicadorconstante();
33
```

```
34          //Lo que hace esta parte de c\'odigo, es que guarda los n\'umeros en un block de notas
35          FileWriter fichero = null;
36          PrintWriter pw = null;
37          try {
38              fichero = new FileWriter("MULTIPLICADOR_CONSTANTE_PSEUDO.txt");
39              pw = new PrintWriter(fichero);
40              System.out.println("Algoritmo de multiplicador constante");
41
42              //Guarda la semilla en la cadena, para calcular la longitud y despu\'es lo convierte a numero
43              System.out.printf("Escribe la semilla:  ");
44              semilla1 = entrada.next();
45              tam1semilla1 = semilla1.length();
46              Nsemilla1 = Integer.parseInt(semilla1);
47
48              //Guarda la constante en la cadena, para calcular la longitud y despu\'es lo convierte a numero
49              System.out.printf("Escribe la constante:  ");
50              constante = entrada.next();
51              tam2constante = constante.length();
52              Nconstante = Integer.parseInt(constante);
53
54              //Aqu\'i inicia el m\'etodo de cuadrados medios para 8 d\'igitos modificado
55              for (i = 1; i <= 30; i++) {
56                  MULTIsemilla = (Nsemilla1 * Nconstante);
57                  multisemilla = Double.toString(MULTIsemilla);
58                  k = a.busca(multisemilla);
59
60                  if (k != -1) {
61                      multisemilla = num.format(MULTIsemilla);
62                  }
63                  tammultisemilla = multisemilla.length() - 2;
64
65                  if (tammultisemilla < 8) {
66                      primerc = (tammultisemilla - tam1semilla1) / 2;
```

```
67                      resultado = multisemilla.substring(primerc, primerc +
    tam1semilla1);
68                      System.out.println(i + ".-     " + resultado);
69                      Nsemilla3 = Double.parseDouble(resultado);
70                      Nr = (Nsemilla3 / 10000.0);
71                      pw.printf("%.4f \n", Nr);
72                      Nsemilla1 = Nsemilla3;
73                      Nconstante = Nconstante;
74                  } else {
75                      primerc = (tammultisemilla - tam1semilla1) / 2;
76                      resultado = multisemilla.substring(primerc, primerc +
    tam1semilla1);
77                      System.out.println(i + ".-     " + resultado);
78                      Nsemilla3 = Double.parseDouble(resultado);
79                      Nr = (Nsemilla3 / 10000.0);
80                      pw.printf("%.4f \n", Nr);
81                      Nsemilla1 = Nsemilla3;
82                      Nconstante = Nconstante;
83                  }
84              }
85          } catch (Exception e) {
86              e.printStackTrace();
87          } finally {
88              try {
89                  if (null != fichero) {
90                      fichero.close();
91                  }
92              } catch (Exception e2) {
93                  e2.printStackTrace();
94              }
95          }
96      }
97  }
```

SECCIÓN A.4

Congruencia lineal mixto

```
1  package congruencialinealmixto;
2
3  import java.io.FileWriter;
4  import java.io.PrintWriter;
5  import java.math.BigDecimal;
6  import java.text.DecimalFormat;
7  import java.util.*;
8
9  /**
10  * @author Anita Ramirez
11  */
12 public class Congruencialinealmixto {
13
14     //En la cadena busca un caracter E
15     public int busca(String cadena) {
16         int n;
17         n = cadena.indexOf("E");
18         return (n);
19     }
20
21     public static void main(String[] args) {
22         Scanner entrada = new Scanner(System.in);
23         Locale.setDefault(Locale.US);
24         DecimalFormat num = new DecimalFormat("#########0");
25
26         //Declaraci\'on de variables
27         String semilla, cmultiplicativa, caditiva, modulo, numero1;
28         int i, k, tamnumero;
29         double numero2 = 0, Nr = 0, Nsemilla, Nconsc, Nconsa, Nmodulo,
       numero;
30
31         Congruencialinealmixto a = new Congruencialinealmixto();
32
33         System.out.println("Algoritmo de congruencial lineal mixto");
```

```
34
35        // Guarda la semilla en una cadena, para saber su longitud y despu\'
      es convertirla en numero
36        System.out.print("Escriba una semilla: ");
37        semilla = entrada.next();
38        Nsemilla = Double.parseDouble(semilla);
39
40        // Guarda la constante c en la cadena, para saber su longitud y d\'
      espues convertirla en numero
41        System.out.print("Escriba una constante aditiva para c: ");
42        caditiva = entrada.next();
43        Nconsc = Double.parseDouble(caditiva);
44
45        // Guarda la constante a en la cadena, para saber su longitud y
      despu\'es convertirla en numero
46        System.out.print("Escriba una constante multiplicativa para a: ");
47        cmultiplicativa = entrada.next();
48        Nconsa = Double.parseDouble(cmultiplicativa);
49
50        // Guarda el m\'odulo en m en la cadena, desea saber su longitud y
      lo convierte a numero
51        System.out.print("Escriba el modulo: ");
52        modulo = entrada.next();
53        Nmodulo = Double.parseDouble(modulo);
54
55        //Lo que hace esta parte de c\'odigo, es que guarda los n\'umeros en
       un block de notas
56        FileWriter fichero = null;
57        PrintWriter pw = null;
58        try {
59            fichero = new FileWriter("CONGRUENCIA_LINEAL_MIXTO_PSEUDO.txt");
60            pw = new PrintWriter(fichero);
61
62            // M\'etodo de congruencia lineal mixto
63            for (i = 1; i <= 100; i++) {
64                numero = ((Nconsa * Nsemilla) + Nconsc) %Nmodulo;
65                numero1 = String.valueOf(numero);
66                k = a.busca(numero1);
```

```
67
68                  if (k != -1) {
69                      numero1 = num.format(numero);
70                      numero = Double.parseDouble(numero1);
71                  }
72                  tamnumero = numero1.length() - 2;
73
74                  // Aqu\'i verifica si est\'an los 4 d\'igitos si no le
    agrega ceros
75                  if (tamnumero == 3) {
76
77                      numero2 = numero / (double) (Nmodulo - 1);
78                      System.out.printf("%d.-     0%.0f    \n", i, numero);
79                  }
80                  if (tamnumero == 2) {
81                      numero2 = numero / (double) (Nmodulo - 1);
82                      System.out.printf("%d.-     00%.0f    \n", i, numero);
83                  }
84                  if (tamnumero == 1) {
85                      numero2 = numero / (double) (Nmodulo - 1);
86                      System.out.printf("%d.-     000%.0f    \n", i, numero);
87                  }
88                  if (tamnumero != 3 && tamnumero != 2 && tamnumero != 1) {
89                      numero2 = (double) numero / (double) (Nmodulo - 1);
90                      System.out.printf("%d.-     %.0f    \n", i, numero);
91                  }
92                  Nr = numero2;
93                  //pw.println(Nr);
94                  pw.printf("%.4f \n", Nr);
95                  Nsemilla = numero;
96              }
97          } catch (Exception e) {
98              e.printStackTrace();
99          } finally {
100             try {
101                 if (null != fichero) {
102                     fichero.close();
103                 }
```

```
104             } catch (Exception e2) {
105                 e2.printStackTrace();
106             }
107         }
108     }
109 }
```

SECCIÓN A.5

Congruencial multiplicativo

```
1  package congruenciamultiplicativo;
2
3  import java.io.FileWriter;
4  import java.io.PrintWriter;
5  import java.math.BigDecimal;
6  import java.text.DecimalFormat;
7  import java.util.*;
8
9  /**
10  * @author Anita Ramirez
11  */
12 public class Congruenciamultiplicativo {
13
14     //En la cadena busca un caracter E
15     public int busca(String cadena) {
16         int n;
17         n = cadena.indexOf("E");
18         return (n);
19     }
20
21     public static void main(String[] args) {
22         Scanner entrada = new Scanner(System.in);
23         Locale.setDefault(Locale.US);
24         DecimalFormat num = new DecimalFormat("########.0");
25
26         //Declaraci\'on de variables
27         String semilla, cmultiplicativa, modulo, numero1;
```

```
28          int i, k, tamnumero;
29          double numero2 = 0, Nr = 0, numero, Nsemilla, Nconsc, Nconsa,
    Nmodulo;
30
31          Congruenciamultiplicativo a = new Congruenciamultiplicativo();
32
33          System.out.println("Algoritmo de congruencial multiplicativo");
34
35          // Guarda la semilla en una cadena, para saber su longitud y despu\'
    es convertirla en numero
36          System.out.print("Escriba una semilla: ");
37          semilla = entrada.next();
38          Nsemilla = Integer.parseInt(semilla);
39
40          // Guarda la constante a en la cadena, para saber su longitud y
    despu\'es convertirla en numero
41          System.out.print("Escriba una constante multiplicativa para a: ");
42          cmultiplicativa = entrada.next();
43          Nconsa = Integer.parseInt(cmultiplicativa);
44
45          // Guarda el m\'odulo en m en la cadena, desea saber su longitud y
    lo convierte a numero
46          System.out.print("Escriba el modulo: ");
47          modulo = entrada.next();
48          Nmodulo = Integer.parseInt(modulo);
49
50          //Lo que hace esta parte de c\'digo, es que guarda los n\'umeros en
    un block de notas
51          FileWriter fichero = null;
52          PrintWriter pw = null;
53          try {
54              fichero = new FileWriter("CONGRUENCIA_MULTIPLICATIVO_PSEUDO.txt"
    );
55              pw = new PrintWriter(fichero);
56
57              // M\'etodo de congruencia multiplicativo
58              for (i = 1; i <= 100; i++) {
59                  numero = (Nconsa * Nsemilla) %Nmodulo;
```

```
60                numero1 = numero1 = String.valueOf(numero);
61                k = a.busca(numero1);
62
63                if (k != -1) {
64                    numero1 = num.format(numero);
65                    numero = Double.parseDouble(numero1);
66
67                }
68                tamnumero = numero1.length() - 2;
69
70                // Aqu\'i verifica si est\'an los 4 d\'igitos si no le
    agrega ceros
71                if (tamnumero == 3) {
72                    numero2 = numero / (double) (Nmodulo - 1);
73                    System.out.printf("%d.-     0%.0f    \n", i, numero);
74                }
75                if (tamnumero == 2) {
76                    numero2 = numero / (double) (Nmodulo - 1);
77                    System.out.printf("%d.-     00%.0f    \n", i, numero);
78                }
79                if (tamnumero == 1) {
80                    numero2 = numero / (double) (Nmodulo - 1);
81                    System.out.printf("%d.-     000%.0f    \n", i, numero);
82                }
83                if (tamnumero != 3 && tamnumero != 2 && tamnumero != 1) {
84                    numero2 = (double) numero / (double) (Nmodulo - 1);
85                    System.out.printf("%d.-     %.0f    \n", i, numero);
86                }
87                Nr = numero2;
88                //pw.println(Nr);
89                pw.printf("%.4f \n", Nr);
90                Nsemilla = numero;
91            }
92        } catch (Exception e) {
93            e.printStackTrace();
94        } finally {
95            try {
96                if (null != fichero) {
```

```
97                      fichero.close();
98                  }
99              } catch (Exception e2) {
100                 e2.printStackTrace();
101             }
102         }
103     }
104 }
```

SECCIÓN A.6

Congruencial no lineal cuadrático

```
1  package congruencialnolinealcuadratico;
2
3  import java.io.FileWriter;
4  import java.io.PrintWriter;
5  import java.math.BigDecimal;
6  import java.text.DecimalFormat;
7  import java.util.*;
8
9  /**
10  * @author Anita Ramirez
11  */
12 public class Congruencialnolinealcuadratico {
13
14     //En la cadena busca un caracter E
15     public int busca(String cadena) {
16         int n;
17         n = cadena.indexOf("E");
18         return (n);
19     }
20
21     public static void main(String[] args) {
22         Scanner entrada = new Scanner(System.in);
23         Locale.setDefault(Locale.US);
24         DecimalFormat num = new DecimalFormat("########0");
25
```

```
26          //Declaraci\'on de variables
27          String semilla, cmultiplicativa, caditiva, cmultiplicativacuadratico
    , modulo, numero1;
28          int i, k, tamnumero;
29          double numero2 = 0, Nr = 0, numero, Nsemilla, Nconsa, Nconsb, Nconsc
    , Nmodulo;
30
31          Congruencialnolinealcuadratico a = new
    Congruencialnolinealcuadratico();
32
33          System.out.println("Algoritmo de congruencial no lineal cuadratico")
    ;
34
35          // Guarda la semilla en una cadena, para saber su longitud y despu\'
    es convertirla en numero
36          System.out.print("Escriba una semilla: ");
37          semilla = entrada.next();
38          Nsemilla = Double.parseDouble(semilla);
39
40          // Guarda la constante a en la cadena, para saber su longitud y
    despu\'es convertirla en numero
41          System.out.print("Escriba una constante multiplicativa para a: ");
42          cmultiplicativacuadratico = entrada.next();
43          Nconsa = Double.parseDouble(cmultiplicativacuadratico);
44
45          // Guarda la constante b en la cadena, para saber su longitud y
    despu\'es convertirla en numero
46          System.out.print("Escriba una constante multiplicativa para b: ");
47          cmultiplicativa = entrada.next();
48          Nconsb = Double.parseDouble(cmultiplicativa);
49
50          // Guarda la constante c en la cadena, para saber su longitud y
    despu\'es convertirla en numero
51          System.out.print("Escriba una constante aditiva para c: ");
52          caditiva = entrada.next();
53          Nconsc = Double.parseDouble(caditiva);
54
```

```
55          // Guarda el m\'odulo en m en la cadena, desea saber su longitud y
    lo convierte a numero
56          System.out.print("Escriba el modulo: ");
57          modulo = entrada.next();
58          Nmodulo = Double.parseDouble(modulo);
59
60          //Lo que hace esta parte de c\'odigo, es que guarda los n\'umeros en
     un block de notas
61          FileWriter fichero = null;
62          PrintWriter pw = null;
63          try {
64              fichero = new FileWriter("CONGRUENCIA_NO_LINEAL_CUADRATICO.txt")
    ;
65              pw = new PrintWriter(fichero);
66
67              // M\'etodo de congruencia no lineal cuadr\'atico
68              for (i = 1; i <= 100; i++) {
69                  numero = ((Nconsa * (Nsemilla * Nsemilla)) + (Nconsb *
    Nsemilla) + Nconsc) %Nmodulo;
70                  numero1 = String.valueOf(numero);
71                  k = a.busca(numero1);
72
73                  if (k != -1) {
74                      numero1 = num.format(numero);
75                      numero = Double.parseDouble(numero1);
76                  }
77                  tamnumero = numero1.length() - 2;
78
79                  // Aqu\'i verifica si est\'an los 4 d\'igitos si no le
    agrega ceros
80                  if (tamnumero == 3) {
81
82                      numero2 = numero / (double) (Nmodulo - 1);
83                      System.out.printf("%d.-     0%.0f    \n", i, numero);
84                  }
85                  if (tamnumero == 2) {
86                      numero2 = numero / (double) (Nmodulo - 1);
87                      System.out.printf("%d.-     00%.0f    \n", i, numero);
```

```
88                  }
89                  if (tamnumero == 1) {
90                      numero2 = numero / (double) (Nmodulo - 1);
91                      System.out.printf("%d.-      000%.0f    \n", i, numero);
92                  }
93                  if (tamnumero != 3 && tamnumero != 2 && tamnumero != 1) {
94                      numero2 = (double) numero / (double) (Nmodulo - 1);
95                      System.out.printf("%d.-      %.0f    \n", i, numero);
96                  }
97                  Nr = numero2;
98                  //pw.println(Nr);
99                  pw.printf("%.4f \n", Nr);
100                 Nsemilla = numero;
101             }
102         } catch (Exception e) {
103             e.printStackTrace();
104         } finally {
105             try {
106                 if (null != fichero) {
107                     fichero.close();
108                 }
109             } catch (Exception e2) {
110                 e2.printStackTrace();
111             }
112         }
113     }
114 }
```

APÉNDICE B

CÓDIGOS DE LOS GENERADORES DE 8 DÍGITOS

Los códigos que a continuación se presentan, se basan en los códigos para los algoritmos generadores de números de cuatro dígitos, dichas codificaciones fueron adaptadas para que se generen números pseudoaleatorios de ocho dígitos con los cuales se trabajó en este proyecto.

SECCIÓN B.1

Cuadrados medios

```
1  package cuadradosmedios;
2
3  import java.io.PrintWriter;
4  import java.io.FileWriter;
5  import java.util.*;
6  import java.text.*;
7  import java.io.*;
8
9  /**
10  * @author Anita Ramirez
11  */
```

```
12 public class Cuadradosmedios {
13
14     //En la cadena busca un caracter E
15     public int busca(String cadena) {
16         int n;
17         n = cadena.indexOf("E");
18         return (n);
19     }
20
21     public static void main(String[] args) {
22         Scanner entrada = new Scanner(System.in);
23         Locale.setDefault(Locale.US);
24         DecimalFormat num = new DecimalFormat("####.0");
25
26         //Declaraci\'on de variables
27         String semilla, cuadsemilla, resultado;
28         int tam2semilla, tam1semilla, i, primerc, k;
29         double numero1semilla, numero2semilla;
30         double Nr = 0;
31
32         Cuadradosmedios a = new Cuadradosmedios();
33
34         //Lo que hace esta parte de c\'odigo, es que guarda los n\'umeros en
    un block de notas
35         FileWriter fichero = null;
36         PrintWriter pw = null;
37         try {
38             fichero = new FileWriter("CUADRADOS_MEDIOS_PSEUDO.txt");
39             pw = new PrintWriter(fichero);
40             System.out.println("Algoritmo de cuadrados medios");
41
42             //Guarda la semilla en la cadena, para calcular la longitud y
   despu\'es lo convierte a numero
43             System.out.print("Escriba semilla:  ");
44             semilla = entrada.next();
45             tam1semilla = semilla.length();
46             numero1semilla = Integer.parseInt(semilla);
47
```

```
48            //Aqu\'i inicia el m\'etodo de cuadrados medios para 8 d\'igitos
    modificado
49            for (i = 1; i <= 10000; i++) {
50                numero2semilla = Math.pow(numero1semilla, 2);
51                cuadsemilla = Double.toString(numero2semilla);
52                k = a.busca(cuadsemilla);
53
54                if (k != -1) {
55                    cuadsemilla = num.format(numero2semilla);
56                }
57                tam2semilla = cuadsemilla.length() - 2;
58
59                if (tam2semilla < 16) {
60                    primerc = (tam2semilla - tam1semilla) / 2;
61                    resultado = cuadsemilla.substring(primerc, primerc +
    tam1semilla);
62                    System.out.println(i + ".-     " + resultado);
63                    numero1semilla = Double.parseDouble(resultado);
64                    Nr = (numero1semilla / 100000000.0);
65                    pw.printf("%.8f \n", Nr);
66                    numero2semilla = numero1semilla;
67                } else {
68                    primerc = (tam2semilla - tam1semilla) / 2;
69                    resultado = cuadsemilla.substring(primerc, primerc +
    tam1semilla);
70                    System.out.println(i + ".-     " + resultado);
71                    numero1semilla = Double.parseDouble(resultado);
72                    Nr = (numero1semilla / 100000000.0);
73                    pw.printf("%.8f \n", Nr);
74                    numero2semilla = numero1semilla;
75                }
76            }
77        } catch (Exception e) {
78            e.printStackTrace();
79        } finally {
80            try {
81                if (null != fichero) {
82                    fichero.close();
```

```
                }
            } catch (Exception e2) {
                e2.printStackTrace();
            }
        }
    }
}
```

SECCIÓN B.2

Productos medios

```
package productosmedios;

import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.*;
import java.text.*;
import java.io.*;

/**
 * @author Anita Ramirez
 */
public class Productosmedios {

    //En la cadena busca un caracter E
    public int busca(String cadena) {
        int n;
        n = cadena.indexOf("E");
        return (n);
    }

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        Locale.setDefault(Locale.US);
        DecimalFormat num = new DecimalFormat("####.0");

        //Declaraci\'on de variables
```

```
27          String semilla1, semilla2, prodsemilla, resultado;
28          int tam1semilla1, tam2semilla2, tamprodsemilla, i, primerc, k;
29          double Nsemilla1, Nsemilla2, Nsemilla3, PROsemilla;
30          double Nr = 0;
31
32          Productosmedios a = new Productosmedios();
33
34          //Lo que hace esta parte de c\'odigo, es que guarda los n\'umeros en
     un block de notas
35          FileWriter fichero = null;
36          PrintWriter pw = null;
37          try {
38              fichero = new FileWriter("PRODUCTOS_MEDIOS_PSEUDO.txt");
39              pw = new PrintWriter(fichero);
40              System.out.println("Algoritmo de productos medios");
41
42              //Guarda la semilla 1 en la cadena 1, para calcular la longitud
    y despu\'es lo convierte a numero 1
43              System.out.print("Escribe la semilla 1:  ");
44              semilla1 = entrada.next();
45              tam1semilla1 = semilla1.length();
46              Nsemilla1 = Integer.parseInt(semilla1);
47
48              //Guarda la semilla 1 en la cadena 1, para calcular la longitud
    y despu\'es lo convierte a numero 1
49              System.out.print("Escribe la semilla 2:  ");
50              semilla2 = entrada.next();
51              tam2semilla2 = semilla2.length();
52              Nsemilla2 = Integer.parseInt(semilla2);
53
54              //Aqu\'i inicia el m\'etodo de cuadrados medios para 8 d\'igitos
     modificado
55              for (i = 1; i <= 41350; i++) {
56                  PROsemilla = (Nsemilla1 * Nsemilla2);
57                  prodsemilla = Double.toString(PROsemilla);
58                  k = a.busca(prodsemilla);
59
60                  if (k != -1) {
```

```
61                  prodsemilla = num.format(PROsemilla);
62              }
63              tamprodsemilla = prodsemilla.length() - 2;
64
65              if (tamprodsemilla < 16) {
66                  primerc = (tamprodsemilla - tam1semilla1) / 2;
67                  resultado = prodsemilla.substring(primerc, primerc +
    tam1semilla1);
68                  System.out.println(i + ".-     " + resultado);
69                  Nsemilla3 = Double.parseDouble(resultado);
70                  Nr = (Nsemilla3 / 100000000);
71                  pw.printf("%.8f \n", Nr);
72                  Nsemilla1 = Nsemilla2;
73                  Nsemilla2 = Nsemilla3;
74              } else {
75                  primerc = (tamprodsemilla - tam1semilla1) / 2;
76                  resultado = prodsemilla.substring(primerc, primerc +
    tam1semilla1);
77                  System.out.println(i + ".-     " + resultado);
78                  Nsemilla3 = Double.parseDouble(resultado);
79                  Nr = (Nsemilla3 / 100000000.0);
80                  pw.printf("%.8f \n", Nr);
81                  Nsemilla1 = Nsemilla2;
82                  Nsemilla2 = Nsemilla3;
83              }
84          }
85      } catch (Exception e) {
86          e.printStackTrace();
87      } finally {
88          try {
89              if (null != fichero) {
90                  fichero.close();
91              }
92          } catch (Exception e2) {
93              e2.printStackTrace();
94          }
95      }
96  }
```

```
97 }
```

SECCIÓN B.3

Multiplicador constante

```
1  package multiplicadorconstante;
2
3  import java.io.FileWriter;
4  import java.io.PrintWriter;
5  import java.util.*;
6  import java.text.*;
7  import java.io.*;
8
9  /**
10  * @author Anita Ramirez
11  */
12 public class Multiplicadorconstante {
13
14     //En la cadena busca un caracter E
15     public int busca(String cadena) {
16         int n;
17         n = cadena.indexOf("E");
18         return (n);
19     }
20
21     public static void main(String[] args) {
22         Scanner entrada = new Scanner(System.in);
23         Locale.setDefault(Locale.US);
24         DecimalFormat num = new DecimalFormat("####.0");
25
26         //Declaraci\'on de variables
27         String semilla1, constante, multisemilla, resultado;
28         int tam1semilla1, tam2constante, tammultisemilla, i, primerc, k;
29         double Nsemilla1, Nconstante, Nsemilla3, MULTIsemilla;
30         double Nr = 0;
31
32         Multiplicadorconstante a = new Multiplicadorconstante();
```

```
33
34          //Lo que hace esta parte de c\'odigo, es que guarda los n\'umeros en
        un block de notas
35          FileWriter fichero = null;
36          PrintWriter pw = null;
37          try {
38              fichero = new FileWriter("MULTIPLICADOR_CONSTANTE_PSEUDO.txt");
39              pw = new PrintWriter(fichero);
40              System.out.println("Algoritmo de multiplicador constante");
41
42              //Guarda la semilla en la cadena, para calcular la longitud y
        despu\'es lo convierte a numero
43              System.out.print("Escribe la semilla:  ");
44              semilla1 = entrada.next();
45              tam1semilla1 = semilla1.length();
46              Nsemilla1 = Integer.parseInt(semilla1);
47
48              //Guarda la constante en la cadena, para calcular la longitud y
        despu\'es lo convierte a numero
49              System.out.print("Escribe la constante:  ");
50              constante = entrada.next();
51              tam2constante = constante.length();
52              Nconstante = Integer.parseInt(constante);
53
54              //Aqu\'i inicia el m\'etodo de cuadrados medios para 8 d\'igitos
         modificado
55              for (i = 1; i <= 15779; i++) {
56                  MULTIsemilla = (Nsemilla1 * Nconstante);
57                  multisemilla = Double.toString(MULTIsemilla);
58                  k = a.busca(multisemilla);
59
60                  if (k != -1) {
61                      multisemilla = num.format(MULTIsemilla);
62                  }
63                  tammultisemilla = multisemilla.length() - 2;
64
65                  if (tammultisemilla < 16) {
66                      primerc = (tammultisemilla - tam1semilla1) / 2;
```

```
67                      resultado = multisemilla.substring(primerc, primerc +
    tam1semilla1);
68                      System.out.println(i + ".-     " + resultado);
69                      Nsemilla3 = Double.parseDouble(resultado);
70                      Nr = (Nsemilla3 / 100000000.0);
71                      pw.printf("%.8f \n", Nr);
72                      Nsemilla1 = Nsemilla3;
73                      Nconstante = Nconstante;
74                  } else {
75                      primerc = (tammultisemilla - tam1semilla1) / 2;
76                      resultado = multisemilla.substring(primerc, primerc +
    tam1semilla1);
77                      System.out.println(i + ".-     " + resultado);
78                      Nsemilla3 = Double.parseDouble(resultado);
79                      Nr = (Nsemilla3 / 100000000.0);
80                      pw.printf("%.8f \n", Nr);
81                      Nsemilla1 = Nsemilla3;
82                      Nconstante = Nconstante;
83                  }
84              }
85          } catch (Exception e) {
86              e.printStackTrace();
87          } finally {
88              try {
89                  if (null != fichero) {
90                      fichero.close();
91                  }
92              } catch (Exception e2) {
93                  e2.printStackTrace();
94              }
95          }
96      }
97  }
```

SECCIÓN B.4

Congruencia lineal mixto

```
1  package congruencialinealmixto;
2
3  import java.io.FileWriter;
4  import java.io.PrintWriter;
5  import java.math.BigDecimal;
6  import java.text.DecimalFormat;
7  import java.util.*;
8
9  /**
10  * @author Anita Ramirez
11  */
12 public class Congruencialinealmixto {
13
14     //En la cadena busca un caracter E
15     public int busca(String cadena) {
16         int n;
17         n = cadena.indexOf("E");
18         return (n);
19     }
20
21     public static void main(String[] args) {
22         Scanner entrada = new Scanner(System.in);
23         Locale.setDefault(Locale.US);
24         DecimalFormat num = new DecimalFormat("########0");
25
26         //Declaraci\'on de variables
27         String semilla, cmultiplicativa, caditiva, modulo, numero1;
28         int i, k, tamnumero;
29         double numero2 = 0, Nr = 0, Nsemilla, Nconsc, Nconsa, Nmodulo,
       numero;
30
31         Congruencialinealmixto a = new Congruencialinealmixto();
32
33         System.out.println("Algoritmo congruencial lineal mixto");
```

```
34
35        // Guarda la semilla en una cadena, para saber su longitud y despu\'
    es convertirla en numero
36        System.out.print("Escriba una semilla: ");
37        semilla = entrada.next();
38        Nsemilla = Double.parseDouble(semilla);
39
40        // Guarda la constante c en la cadena, para saber su longitud y
    despu\'es convertirla en numero
41        System.out.print("Escriba una constante aditiva para c: ");
42        caditiva = entrada.next();
43        Nconsc = Double.parseDouble(caditiva);
44
45        // Guarda la constante a en la cadena, para saber su longitud y
    despu\'es convertirla en numero
46        System.out.print("Escriba una constante multiplicativa para a: ");
47        cmultiplicativa = entrada.next();
48        Nconsa = Double.parseDouble(cmultiplicativa);
49
50        // Guarda el m\'odulo en m en la cadena, desea saber su longitud y
    lo convierte a numero
51        System.out.print("Escriba el m\'odulo: ");
52        modulo = entrada.next();
53        Nmodulo = Double.parseDouble(modulo);
54
55        //Lo que hace esta parte de c\'odigo, es que guarda los n\'umeros en
     un block de notas
56        FileWriter fichero = null;
57        PrintWriter pw = null;
58        try {
59            fichero = new FileWriter("CONGRUENCIA_LINEAL_MIXTO_PSEUDO.txt");
60            pw = new PrintWriter(fichero);
61
62            // M\'etodo de congruencia lineal mixto
63            for (i = 1; i <= 500000; i++) {
64                numero = ((Nconsa * Nsemilla) + Nconsc) %Nmodulo;
65                numero1 = String.valueOf(numero);
66                k = a.busca(numero1);
```

```
67
68                  if (k != -1) {
69                      numero1 = num.format(numero);
70                      numero = Double.parseDouble(numero1);
71                  }
72                  tamnumero = numero1.length() - 2;
73
74                  // Aqu\'i verifica si est\'an los 8 d\'igitos si no le
    agrega ceros
75                  if (tamnumero == 7) {
76
77                      numero2 = numero / (double) (Nmodulo - 1);
78                      System.out.printf("%d.-     0%.0f    \n", i, numero);
79                  }
80                  if (tamnumero == 6) {
81                      numero2 = numero / (double) (Nmodulo - 1);
82                      System.out.printf("%d.-     00%.0f    \n", i, numero);
83                  }
84                  if (tamnumero == 5) {
85                      numero2 = numero / (double) (Nmodulo - 1);
86                      System.out.printf("%d.-     000%.0f    \n", i, numero);
87                  }
88                  if (tamnumero != 7 && tamnumero != 6 && tamnumero != 5) {
89                      numero2 = (double) numero / (double) (Nmodulo - 1);
90                      System.out.printf("%d.-     %.0f    \n", i, numero);
91                  }
92                  Nr = numero2;
93                  pw.printf("%.8f \n", Nr);
94                  Nsemilla = numero;
95              }
96          } catch (Exception e) {
97              e.printStackTrace();
98          } finally {
99              try {
100                 if (null != fichero) {
101                     fichero.close();
102                 }
103             } catch (Exception e2) {
```

```
                e2.printStackTrace();
            }
        }
    }
}
```

SECCIÓN B.5

Congruencial multiplicativo

```
package congruencialmultiplicativo;

import java.io.FileWriter;
import java.io.PrintWriter;
import java.math.BigDecimal;
import java.text.DecimalFormat;
import java.util.*;

/**
 * @author Anita Ramirez
 */
public class Congruencialmultiplicativo {

    //En la cadena busca un caracter E
    public int busca(String cadena) {
        int n;
        n = cadena.indexOf("E");
        return (n);
    }

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        Locale.setDefault(Locale.US);
        DecimalFormat num = new DecimalFormat("########0");

        //Declaraci\'on de variables
        String semilla, cmultiplicativa, modulo, numero1;
        int i, k, tamnumero;
```

```
29        double numero2 = 0, Nr = 0, numero, Nsemilla, Nconsc, Nconsa,
    Nmodulo;
30
31        Congruencialmultiplicativo a = new Congruencialmultiplicativo();
32
33        System.out.println("Algoritmo congruencial lineal multiplicativo");
34
35        // Guarda la semilla en una cadena, para saber su longitud y despu\'
    es convertirla en numero
36        System.out.print("Escriba una semilla: ");
37        semilla = entrada.next();
38        Nsemilla = Double.parseDouble(semilla);
39
40        // Guarda la constante a en la cadena, para saber su longitud y
    despu\'es convertirla en numero
41        System.out.print("Escriba una constante multiplicativa para a: ");
42        cmultiplicativa = entrada.next();
43        Nconsa = Double.parseDouble(cmultiplicativa);
44
45        // Guarda el m\'odulo en m en la cadena, desea saber su longitud y
    lo convierte a numero
46        System.out.print("Escriba el m\'odulo: ");
47        modulo = entrada.next();
48        Nmodulo = Double.parseDouble(modulo);
49
50        //Lo que hace esta parte de c\'odigo, es que guarda los n\'umeros en
     un block de notas
51        FileWriter fichero = null;
52        PrintWriter pw = null;
53        try {
54            fichero = new FileWriter("CONGRUENCIAL_MULTIPLICATIVO_PSEUDO.txt
    ");
55            pw = new PrintWriter(fichero);
56
57            // M\'etodo de congruencia lineal mixto
58            for (i = 1; i <= 256999; i++) {
59                numero = (Nconsa * Nsemilla) %Nmodulo;
60                numero1 = numero1 = String.valueOf(numero);
```

```
61                k = a.busca(numero1);
62
63                if (k != -1) {
64                    numero1 = num.format(numero);
65                    numero = Double.parseDouble(numero1);
66                }
67                tamnumero = numero1.length() - 2;
68
69                // Aqu\'i verifica si est\'an los 8 d\'igitos si no le
    agrega ceros
70                if (tamnumero == 7) {
71                    numero2 = numero / (double) (Nmodulo - 1);
72                    System.out.printf("%d.-     0%.0f    \n", i, numero);
73                }
74                if (tamnumero == 6) {
75                    numero2 = numero / (double) (Nmodulo - 1);
76                    System.out.printf("%d.-     00%.0f    \n", i, numero);
77                }
78                if (tamnumero == 5) {
79                    numero2 = numero / (double) (Nmodulo - 1);
80                    System.out.printf("%d.-     000%.0f    \n", i, numero);
81                }
82                if (tamnumero != 7 && tamnumero != 6 && tamnumero != 5) {
83                    numero2 = (double) numero / (double) (Nmodulo - 1);
84                    System.out.printf("%d.-     %.0f    \n", i, numero);
85                }
86                Nr = numero2;
87                pw.printf("%.8f \n", Nr);
88                Nsemilla = numero;
89            }
90        } catch (Exception e) {
91            e.printStackTrace();
92        } finally {
93            try {
94                if (null != fichero) {
95                    fichero.close();
96                }
97            } catch (Exception e2) {
```

```
98                  e2.printStackTrace();
99              }
100         }
101     }
102 }
```

SECCIÓN B.6

Congruencial no lineal cuadrático

```
1  package congruencialnolinealcuadratico;
2
3  import java.io.FileWriter;
4  import java.io.PrintWriter;
5  import java.math.BigDecimal;
6  import java.text.DecimalFormat;
7  import java.util.*;
8
9  /**
10  * @author Anita Ramirez
11  */
12 public class Congruencialnolinealcuadratico {
13
14     //En la cadena busca un caracter E
15     public int busca(String cadena) {
16         int n;
17         n = cadena.indexOf("E");
18         return (n);
19     }
20
21     public static void main(String[] args) {
22         Scanner entrada = new Scanner(System.in);
23         Locale.setDefault(Locale.US);
24         DecimalFormat num = new DecimalFormat("########.0");
25
26         //Declaraci\'on de variables
27         String semilla, cmultiplicativa, caditiva, cmultiplicativacuadratico
    , modulo, numero1;
```

```
28        int i, k, tamnumero;
29        double numero2 = 0, Nr = 0, numero, Nsemilla, Nconsa, Nconsb, Nconsc
      , Nmodulo;
30
31        Congruencialnolinealcuadratico a = new
      Congruencialnolinealcuadratico();
32
33        System.out.println("Algoritmo congruencial no lineal cuadr\'atico");
34
35        // Guarda la semilla en una cadena, para saber su longitud y despu\'
      es convertirla en numero
36        System.out.print("Escriba una semilla: ");
37        semilla = entrada.next();
38        Nsemilla = Double.parseDouble(semilla);
39
40        // Guarda la constante a en la cadena, para saber su longitud y
      despu\'es convertirla en numero
41        System.out.print("Escriba una constante multiplicativa para a: ");
42        cmultiplicativacuadratico = entrada.next();
43        Nconsa = Double.parseDouble(cmultiplicativacuadratico);
44
45        // Guarda la constante b en la cadena, para saber su longitud y
      despu\'es convertirla en numero
46        System.out.print("Escriba una constante multiplicativa para b: ");
47        cmultiplicativa = entrada.next();
48        Nconsb = Double.parseDouble(cmultiplicativa);
49
50        // Guarda la constante c en la cadena, para saber su longitud y
      despu\'es convertirla en numero
51        System.out.print("Escriba una constante aditiva para c: ");
52        caditiva = entrada.next();
53        Nconsc = Double.parseDouble(caditiva);
54
55        // Guarda el m\'odulo en m en la cadena, desea saber su longitud y
      lo convierte a numero
56        System.out.print("Escriba el m\'odulo: ");
57        modulo = entrada.next();
58        Nmodulo = Double.parseDouble(modulo);
```

```
59
60          //Lo que hace esta parte de c\'odigo, es que guarda los n\'umeros en
        un block de notas
61          FileWriter fichero = null;
62          PrintWriter pw = null;
63          try {
64              fichero = new FileWriter("CONGRUENCIA_NO_LINEAL_CUADRATICO.txt")
    ;
65              pw = new PrintWriter(fichero);
66
67              // M\'etodo de congruencia no lineal cuadratico
68              for (i = 1; i <= 5000; i++) {
69                  numero = ((Nconsa * (Nsemilla * Nsemilla)) + (Nconsb *
    Nsemilla) + Nconsc) %Nmodulo;
70                  numero1 = String.valueOf(numero);
71                  k = a.busca(numero1);
72
73                  if (k != -1) {
74                      numero1 = num.format(numero);
75                      numero = Double.parseDouble(numero1);
76                  }
77                  tamnumero = numero1.length() - 2;
78
79                  // Aqu\'i verifica si est\'an los 4 d\'igitos si no le
    agrega ceros
80                  if (tamnumero == 7) {
81
82                      numero2 = numero / (double) (Nmodulo - 1);
83                      System.out.printf("%d.-     0%.0f    \n", i, numero);
84                  }
85                  if (tamnumero == 6) {
86                      numero2 = numero / (double) (Nmodulo - 1);
87                      System.out.printf("%d.-     00%.0f    \n", i, numero);
88                  }
89                  if (tamnumero == 5) {
90                      numero2 = numero / (double) (Nmodulo - 1);
91                      System.out.printf("%d.-     000%.0f    \n", i, numero);
92                  }
```

```
93                  if (tamnumero != 7 && tamnumero != 6 && tamnumero != 5) {
94                      numero2 = (double) numero / (double) (Nmodulo - 1);
95                      System.out.printf("%d.-      %.0f    \n", i, numero);
96                  }
97                  Nr = numero2;
98                  pw.printf("%.8f \n", Nr);
99                  Nsemilla = numero;
100             }
101         } catch (Exception e) {
102             e.printStackTrace();
103         } finally {
104             try {
105                 if (null != fichero) {
106                     fichero.close();
107                 }
108             } catch (Exception e2) {
109                 e2.printStackTrace();
110             }
111         }
112     }
113 }
```

APÉNDICE C

PRUEBAS DE ALEATORIEDAD EN CÓDIGO R

Este código, es para hacer las pruebas de aleatoriedad mediante el software R Studio, y así ejecutarlo las veces que sea necesario.

SECCIÓN C.1

Prueba de medias

```
# PRUEBA DE MEDIAS
# Se acepta la hip\'otesis si la media de los pseudoaleatorios esta dentro
    del intervalo
# [lim.inf.pseudo , lim.sup.pseudo] o es igual a 0.5
# si no se rechaza

pseudoaleatorios<-scan("D:/Users/Anita Ramirez/Documents/DESARROLLANDO TESIS
    /PROGRAMAS 8 DIGITOS/NO CONGRUENCIALES/CUADRADOSMEDIOS/CUADRADOS_MEDIOS_
    PSEUDO.txt")

n<-length(pseudoaleatorios)

alfa=0.05;
```

```
z<-abs(qnorm(alfa/2))
z

# Intervalos
lim.inf.pseudo<-0.5-(z/sqrt(12*n))
lim.inf.pseudo

lim.sup.pseudo<-0.5+(z/sqrt(12*n))
lim.sup.pseudo

# Media de los pseudoaleatorios
media.pseudo<-mean(pseudoaleatorios)
media.pseudo
```

SECCIÓN C.2

Prueba de varianza

```
# PRUEBA DE VARIANZA
# Se acepta la hip\'otesis si la varianza de los pseudoaleatorios esta
    dentro
# del intervalo [lim.inf.var , lim.sup.var] o es igual a 1/12
# si no se rechaza

#Chi de las tablas(R)
chi.inf<-qchisq((1-(alfa/2)),n-1,lower.tail=FALSE)
chi.inf

chi.sup<-qchisq((alfa/2),n-1,lower.tail=FALSE)
chi.sup

#Intervalos
lim.inf.var<-(chi.inf/(12*n-1))
lim.inf.var

lim.sup.var<-(chi.sup/(12*n-1))
lim.sup.var

```

```
#varianza de los pseudoaleatorios
var(pseudoaleatorios)
```

SECCIÓN C.3

Prueba de χ^2

```
# PRUEBA DE JI-CUADRADA
# El estad\'istico x_0 debe ser menor al estad\'istico correspondiente de (
    x0^2)_(alfa,n-1)
# si el estad\'istico es mayor al (x0^2)_(alfa,n-1) entonces se rechaza.

# Calculamos el n\'umero de clases usando la f\'ormula de Sturge
k1 <- nclass.Sturges(pseudoaleatorios)

# Formamos los intervalos de clase
intervalos <- cut(pseudoaleatorios, breaks = k1)

# Tabla de frecuencias absolutas
tabla <- as.data.frame(table(intervalos))

# Total de elementos
Total<-sum(tabla$Freq)

# Valor esperado
e=n/k1

# Frecuencia esperada
Ei<-rep(e, times = k1)

# Renombrar las frecuencias observada
Oi<-c(tabla$Freq)

# Frecuencias para el estad\'istico
ecu<-c((Ei-Oi)^2/Ei)

# Arma tabla de frecuencias
resul<- data.frame(tabla$intervalos,Oi,Ei,ecu)
```

```
31
32 # Visualizar tabla
33 View(resul)
34
35 # Estad\'istico (x0^2)_(alfa,n-1)
36 chi.tabla<-qchisq(alfa,k1-1,lower.tail=FALSE)
37 chi.tabla
38
39 # Estad\'istico x_0 de los pseudoaleatorios
40 X_0=sum(ecu)
41 X_0
```

APÉNDICE D

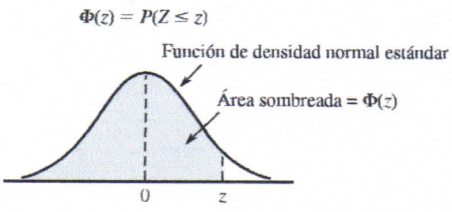
TABLAS DE DISTRIBUCIONES

SECCIÓN D.1

Distribución normal

668 Apéndice/Tablas

Tabla A.3 Áreas de la Curva normal estándar



z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
−3.4	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0002
−3.3	0.0005	0.0005	0.0005	0.0004	0.0004	0.0004	0.0004	0.0004	0.0004	0.0003
−3.2	0.0007	0.0007	0.0006	0.0006	0.0006	0.0006	0.0006	0.0005	0.0005	0.0005
−3.1	0.0010	0.0009	0.0009	0.0009	0.0008	0.0008	0.0008	0.0008	0.0007	0.0007
−3.0	0.0013	0.0013	0.0013	0.0012	0.0012	0.0011	0.0011	0.0011	0.0010	0.0010
−2.9	0.0019	0.0018	0.0017	0.0017	0.0016	0.0016	0.0015	0.0015	0.0014	0.0014
−2.8	0.0026	0.0025	0.0024	0.0023	0.0023	0.0022	0.0021	0.0021	0.0020	0.0019
−2.7	0.0035	0.0034	0.0033	0.0032	0.0031	0.0030	0.0029	0.0028	0.0027	0.0026
−2.6	0.0047	0.0045	0.0044	0.0043	0.0041	0.0040	0.0039	0.0038	0.0037	0.0036
−2.5	0.0062	0.0060	0.0059	0.0057	0.0055	0.0054	0.0052	0.0051	0.0049	0.0038
−2.4	0.0082	0.0080	0.0078	0.0075	0.0073	0.0071	0.0069	0.0068	0.0066	0.0064
−2.3	0.0107	0.0104	0.0102	0.0099	0.0096	0.0094	0.0091	0.0089	0.0087	0.0084
−2.2	0.0139	0.0136	0.0132	0.0129	0.0125	0.0122	0.0119	0.0116	0.0113	0.0110
−2.1	0.0179	0.0174	0.0170	0.0166	0.0162	0.0158	0.0154	0.0150	0.0146	0.0143
−2.0	0.0228	0.0222	0.0217	0.0212	0.0207	0.0202	0.0197	0.0192	0.0188	0.0183

−1.9	0.0287	0.0281	0.0274	0.0268	0.0262	0.0256	0.0250	0.0244	0.0239	0.0233
−1.8	0.0359	0.0352	0.0344	0.0336	0.0329	0.0322	0.0314	0.0307	0.0301	0.0294
−1.7	0.0446	0.0436	0.0427	0.0418	0.0409	0.0401	0.0392	0.0384	0.0375	0.0367
−1.6	0.0548	0.0537	0.0526	0.0516	0.0505	0.0495	0.0485	0.0475	0.0465	0.0455
−1.5	0.0668	0.0655	0.0643	0.0630	0.0618	0.0606	0.0594	0.0582	0.0571	0.0559
−1.4	0.0808	0.0793	0.0778	0.0764	0.0749	0.0735	0.0722	0.0708	0.0694	0.0681
−1.3	0.0968	0.0951	0.0934	0.0918	0.0901	0.0885	0.0869	0.0853	0.0838	0.0823
−1.2	0.1151	0.1131	0.1112	0.1093	0.1075	0.1056	0.1038	0.1020	0.1003	0.0985
−1.1	0.1357	0.1335	0.1314	0.1292	0.1271	0.1251	0.1230	0.1210	0.1190	0.1170
−1.0	0.1587	0.1562	0.1539	0.1515	0.1492	0.1469	0.1446	0.1423	0.1401	0.1379
−0.9	0.1841	0.1814	0.1788	0.1762	0.1736	0.1711	0.1685	0.1660	0.1635	0.1611
−0.8	0.2119	0.2090	0.2061	0.2033	0.2005	0.1977	0.1949	0.1922	0.1894	0.1867
−0.7	0.2420	0.2389	0.2358	0.2327	0.2296	0.2266	0.2236	0.2206	0.2177	0.2148
−0.6	0.2743	0.2709	0.2676	0.2643	0.2611	0.2578	0.2546	0.2514	0.2483	0.2451
−0.5	0.3085	0.3050	0.3015	0.2981	0.2946	0.2912	0.2877	0.2843	0.2810	0.2776
−0.4	0.3446	0.3409	0.3372	0.3336	0.3300	0.3264	0.3228	0.3192	0.3156	0.3121
−0.3	0.3821	0.3783	0.3745	0.3707	0.3669	0.3632	0.3594	0.3557	0.3520	0.3482
−0.2	0.4207	0.4168	0.4129	0.4090	0.4052	0.4013	0.3974	0.3936	0.3897	0.3859
−0.1	0.4602	0.4562	0.4522	0.4483	0.4443	0.4404	0.4364	0.4325	0.4286	0.4247
−0.0	0.5000	0.4960	0.4920	0.4880	0.4840	0.4801	0.4761	0.4721	0.4681	0.4641

(*continúa*)

Tabla D.1: Distribución normal, [5, pág.668]

Tabla A.3 Áreas de la Curva normal estándar *(continuación)*

$\Phi(z) = P(Z \leq z)$

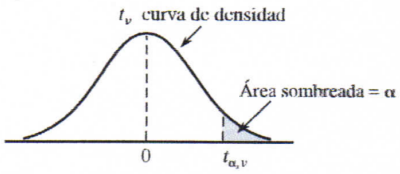
z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9278	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990
3.1	0.9990	0.9991	0.9991	0.9991	0.9992	0.9992	0.9992	0.9992	0.9993	0.9993
3.2	0.9993	0.9993	0.9994	0.9994	0.9994	0.9994	0.9994	0.9995	0.9995	0.9995
3.3	0.9995	0.9995	0.9995	0.9996	0.9996	0.9996	0.9996	0.9996	0.9996	0.9997
3.4	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9997	0.9998

Tabla D.2: Continuación de la distribución normal, [5, pág.669]

SECCIÓN D.2

t-Student

Tabla A.5 Valores críticos para Distribuciones *t*



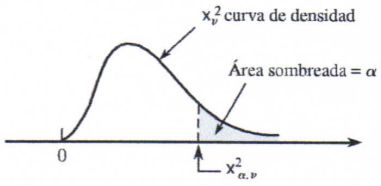
	α						
ν	0.10	0.05	0.025	0.01	0.005	0.001	0.0005
1	3.078	6.314	12.706	31.821	63.657	318.31	636.62
2	1.886	2.920	4.303	6.965	9.925	22.326	31.598
3	1.638	2.353	3.182	4.541	5.841	10.213	12.924
4	1.533	2.132	2.776	3.747	4.604	7.173	8.610
5	1.476	2.015	2.571	3.365	4.032	5.893	6.869
6	1.440	1.943	2.447	3.143	3.707	5.208	5.959
7	1.415	1.895	2.365	2.998	3.499	4.785	5.408
8	1.397	1.860	2.306	2.896	3.355	4.501	5.041
9	1.383	1.833	2.262	2.821	3.250	4.297	4.781
10	1.372	1.812	2.228	2.764	3.169	4.144	4.587
11	1.363	1.796	2.201	2.718	3.106	4.025	4.437
12	1.356	1.782	2.179	2.681	3.055	3.930	4.318
13	1.350	1.771	2.160	2.650	3.012	3.852	4.221
14	1.345	1.761	2.145	2.624	2.977	3.787	4.140
15	1.341	1.753	2.131	2.602	2.947	3.733	4.073
16	1.337	1.746	2.120	2.583	2.921	3.686	4.015
17	1.333	1.740	2.110	2.567	2.898	3.646	3.965
18	1.330	1.734	2.101	2.552	2.878	3.610	3.922
19	1.328	1.729	2.093	2.539	2.861	3.579	3.883
20	1.325	1.725	2.086	2.528	2.845	3.552	3.850
21	1.323	1.721	2.080	2.518	2.831	3.527	3.819
22	1.321	1.717	2.074	2.508	2.819	3.505	3.792
23	1.319	1.714	2.069	2.500	2.807	3.485	3.767
24	1.318	1.711	2.064	2.492	2.797	3.467	3.745
25	1.316	1.708	2.060	2.485	2.787	3.450	3.725
26	1.315	1.706	2.056	2.479	2.779	3.435	3.707
27	1.314	1.703	2.052	2.473	2.771	3.421	3.690
28	1.313	1.701	2.048	2.467	2.763	3.408	3.674
29	1.311	1.699	2.045	2.462	2.756	3.396	3.659
30	1.310	1.697	2.042	2.457	2.750	3.385	3.646
32	1.309	1.694	2.037	2.449	2.738	3.365	3.622
34	1.307	1.691	2.032	2.441	2.728	3.348	3.601
36	1.306	1.688	2.028	2.434	2.719	3.333	3.582
38	1.304	1.686	2.024	2.429	2.712	3.319	3.566
40	1.303	1.684	2.021	2.423	2.704	3.307	3.551
50	1.299	1.676	2.009	2.403	2.678	3.262	3.496
60	1.296	1.671	2.000	2.390	2.660	3.232	3.460
120	1.289	1.658	1.980	2.358	2.617	3.160	3.373
∞	1.282	1.645	1.960	2.326	2.576	3.090	3.291

Tabla D.3: Distribución t, [5, pág.671]

SECCIÓN D.3

χ^2

Tabla A.7 Valores críticos para distribuciones *chi*-cuadrada



	α									
ν	0.995	0.99	0.975	0.95	0.90	0.10	0.05	0.025	0.01	0.005
1	0.000	0.000	0.001	0.004	0.016	2.706	3.843	5.025	6.637	7.882
2	0.010	0.020	0.051	0.103	0.211	4.605	5.992	7.378	9.210	10.597
3	0.072	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.344	12.837
4	0.207	0.297	0.484	0.711	1.064	7.779	9.488	11.143	13.277	14.860
5	0.412	0.554	0.831	1.145	1.610	9.236	11.070	12.832	15.085	16.748
6	0.676	0.872	1.237	1.635	2.204	10.645	12.592	14.440	16.812	18.548
7	0.989	1.239	1.690	2.167	2.833	12.017	14.067	16.012	18.474	20.276
8	1.344	1.646	2.180	2.733	3.490	13.362	15.507	17.534	20.090	21.954
9	1.735	2.088	2.700	3.325	4.168	14.684	16.919	19.022	21.665	23.587
10	2.156	2.558	3.247	3.940	4.865	15.987	18.307	20.483	23.209	25.188
11	2.603	3.053	3.816	4.575	5.578	17.275	19.675	21.920	24.724	26.755
12	3.074	3.571	4.404	5.226	6.304	18.549	21.026	23.337	26.217	28.300
13	3.565	4.107	5.009	5.892	7.041	19.812	22.362	24.735	27.687	29.817
14	4.075	4.660	5.629	6.571	7.790	21.064	23.685	26.119	29.141	31.319
15	4.600	5.229	6.262	7.261	8.547	22.307	24.996	27.488	30.577	32.799
16	5.142	5.812	6.908	7.962	9.312	23.542	26.296	28.845	32.000	34.267
17	5.697	6.407	7.564	8.682	10.085	24.769	27.587	30.190	33.408	35.716
18	6.265	7.015	8.231	9.390	10.865	25.989	28.869	31.526	34.805	37.156
19	6.843	7.632	8.906	10.117	11.651	27.203	30.143	32.852	36.190	38.580
20	7.434	8.260	9.591	10.851	12.443	28.412	31.410	34.170	37.566	39.997
21	8.033	8.897	10.283	11.591	13.240	29.615	32.670	35.478	38.930	41.399
22	8.643	9.542	10.982	12.338	14.042	30.813	33.924	36.781	40.289	42.796
23	9.260	10.195	11.688	13.090	14.848	32.007	35.172	38.075	41.637	44.179
24	9.886	10.856	12.401	13.848	15.659	33.196	36.415	39.364	42.980	45.558
25	10.519	11.523	13.120	14.611	16.473	34.381	37.652	40.646	44.313	46.925
26	11.160	12.198	13.844	15.379	17.292	35.563	38.885	41.923	45.642	48.290
27	11.807	12.878	14.573	16.151	18.114	36.741	40.113	43.194	46.962	49.642
28	12.461	13.565	15.308	16.928	18.939	37.916	41.337	44.461	48.278	50.993
29	13.120	14.256	16.147	17.708	19.768	39.087	42.557	45.772	49.586	52.333
30	13.787	14.954	16.791	18.493	20.599	40.256	43.773	46.979	50.892	53.672
31	14.457	15.655	17.538	19.280	21.433	41.422	44.985	48.231	52.190	55.000
32	15.134	16.362	18.291	20.072	22.271	42.585	46.194	49.480	53.486	56.328
33	15.814	17.073	19.046	20.866	23.110	43.745	47.400	50.724	54.774	57.646
34	16.501	17.789	19.806	21.664	23.952	44.903	48.602	51.966	56.061	58.964
35	17.191	18.508	20.569	22.465	24.796	46.059	49.802	53.203	57.340	60.272
36	17.887	19.233	21.336	23.269	25.643	47.212	50.998	54.437	58.619	61.581
37	18.584	19.960	22.105	24.075	26.492	48.363	52.192	55.667	59.891	62.880
38	19.289	20.691	22.878	24.884	27.343	49.513	53.384	56.896	61.162	64.181
39	19.994	21.425	23.654	25.695	28.196	50.660	54.572	58.119	62.426	65.473
40	20.706	22.164	24.433	26.509	29.050	51.805	55.758	59.342	63.691	66.766

Para $\nu > 40$, $\chi^2_{\alpha,\nu} \approx \nu\left(1 - \frac{2}{9\nu} + z_\alpha\sqrt{\frac{2}{9\nu}}\right)^3$

Tabla D.4: Distribución χ^2, [5, pág.673]

APÉNDICE E

CÓDIGO DE LA PÁGINA WEB

SECCIÓN E.1

home.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.
    org">

<head>
<meta charset="utf-8">
<title>Mi sitio web</title>
<link href="css/Style_web.css" rel="stylesheet" type="text/css">
</head>

<body>
<!--INICIO DEL SHADOW-->
    <div id="shadow"></div>
<!--INICIO DEL CONTAINER-->
<div id="container">

      <div id="header">
        <img src="web_img/img_header.jpg" width="960" height="191">
      </div> <!--FIN DEL HEADER-->
```

```
19
20      <div id="nav">
21       <ul>
22      <li><a th:href="@{/}" name="Inicio" id="home.html">Inicio</a></li>
23      <li><a th:href="@{/historia}" name="Historia" id="historia.html">
24      Historia</a></li>
25      <li><a th:href="@{/publico}" name="Publico" id="publico.html">
26      Publico</a></li>
27      <li><a th:href="@{/privado}" name="Privado" id="privado.html">
28      Privado</a></li>
29       </ul>
30       </div>
31       <!--FIN DEL NAV-->
32
33       <div id="article">
34
35       <div id="section">
36       <h1> Bienvenidos </h1>
37
38      <h2>Universidad Aut\'onoma de la Ciudad de M\'exico<br>
39      Ana Karen Ram\'irez L\'opez <br>
40      Estudiante de la Lic. en Modelaci\'on Matem\'atica<br>
41      Director: M en C. Juan Carlos Aguilar Franco<br>
42      <br><br>
43      Proyecto de tesis<br>
44      Modelo de un token de ocho d\'igitos y pruebas de aleatoriedad ,<br>
45      para n\'umeros pseudoaleatorios.</h2>
46      </div><!--FIN DEL SECTION-->
47
48     <div class="slider">
49   <ul>
50   <li><img src="web_img/san_lorenzo1.jpg" width="492" height="333" alt="">
51   </li>
52     <li><img src="web_img/san_lorenzo2.jpg" width="492" height="333" alt="">
53     </li>
54     <li><img src="web_img/san_lorenzo3.jpg" width="492" height="333" alt="">
55     </li>
56     <li><img src="web_img/san_lorenzo4.jpg" width="492" height="333" alt="">
```

```
57      </li>
58    </ul>
59      <br>
60    </div> <!--FIN DEL SLIDER-->
61
62      </div><!--FIN DEL ARTICLE-->
63
64        <div id="aside">
65        <h3>
66        <br><br><br>
67        Universidad Aut\'onoma de la Ciudad de M\'exico <br><br>
68        San Lorenzo Tezonco <br><br>
69        Calle Prolongaci\'on San Isidro No. 151, <br>
70        Col. San Lorenzo Tezonco, <br>
71        Del. Iztapalapa, M\'exico, CDMX, <br>
72        C.P. 09790. <br>
73        <br><br><br>
74        <h3><a href="https://www.uacm.edu.mx/">Sitio de la UACM</a></h3>
75        </h3>
76        </div> <!--FIN DEL ASIDE-->
77
78        <div id="footer">
79         <h5>
80         \copyright2020 <br>
81         P\'agina personal <br>
82         Ana Karen Ram\'irez L\'opez<br>
83         </h5>
84        </div> <!--FIN DEL FOOTER-->
85      </div><!--FIN DEL CONTAINER-->
86     </div><!--FIN DEL SHADOW-->
87  </body>
88  </html>
```

SECCIÓN E.2

historia.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.
    org">

<head>
<meta charset="utf-8">
<title>Mi sitio web</title>
<link href="css/Style_web.css" rel="stylesheet" type="text/css">
</head>

<body>
<!--INICIO DEL SHADOW-->
    <div id="shadow"></div>
<!--INICIO DEL CONTAINER-->
<div id="container">

      <div id="header">
        <img src="web_img/img_header.jpg" width="960" height="191">
      </div> <!--FIN DEL HEADER-->

      <div id="nav">
      <ul>
     <li><a th:href="@{/}" name="Inicio" id="home.html">Inicio</a></li>
     <li><a th:href="@{/historia}" name="Historia" id="historia.html">
     Historia</a></li>
     <li><a th:href="@{/publico}" name="Publico" id="publico.html">
     Publico</a></li>
     <li><a th:href="@{/privado}" name="Privado" id="privado.html">
     Privado</a></li>
      </ul>
      </div>
      <!--FIN DEL NAV-->

     <div id="article">
```

```
34
35      <div id="section">
36      <h1>Antecedentes de los n\'Umeros aleatorios</h1>
37     <h3>
38     <p align=justify>
39      
40     En este apartado se presenta una breve parte de historia, sobre los n\'
       umeros aleatorios que son utilizados, durante el siglo XVII, se
       desarrollaron distintos campos de estudio, uno de los principales e
       importantes fueron los juegos de azar, pues con ellos se representaron y
        simularon fen\'omenos aleatorios. En la d\'ecada de los cuarenta se
       cita, <q>la construcci\'on de modelos arranca desde el renacimiento, el
       uso moderno de la palabra simulaci\'on data de 1940, cuando los cient\'
       ificos Von Neuman y Ulam que trabajaban en el proyecto Montecarlo,
       durante la segunda guerra mundial, resolvieron problemas de reacciones
       nucleares cuya soluci\'on experimental ser\'ia muy cara y el an\'alisis
       matem\'atico demasiado complicado.</q> </p>
41
42     <p align=justify>
43      
44       Al ser complicado la realizaci\'on del an\'alisis matem\'atico, da pie
       para poder crear los n\'umeros pseudoaleatorios
45       y podemos hacer simulaciones para poder manejarlos como uno desea, con
       diferentes par\'ametros, como se requiera.</p>
46
47      <p align=justify>
48      
49       Esto es algo breve de la historia que se encontr\'o, pero podr\'ias
       leer la tesis y ah\'i explica un poco m\'as sobre el
50       comportamiento de los generadores que es de lo que se trabaj\'o.</p>
51      </h3>
52      </div><!--FIN DEL SECTION-->
53
54      </div><!--FIN DEL ARTICLE-->
55
56       <div id="aside">
57       <h3>
58       <br><br><br>
```

```
59        Universidad Aut\'onoma de la Ciudad de M\'exico <br><br>
60        San Lorenzo Tezonco <br><br>
61        Calle Prolongaci\'on San Isidro No. 151, <br>
62        Col. San Lorenzo Tezonco, <br>
63        Del. Iztapalapa, M\'exico, CDMX, <br>
64        C.P. 09790. <br>
65        <br><br><br>
66        <h3><a href="https://www.uacm.edu.mx/">Sitio de la UACM</a></h3>
67        </h3>
68        </div> <!--FIN DEL ASIDE-->
69
70        <div id="footer">
71          <h5>
72          \copyright2020 <br>
73          P\'agina personal <br>
74          Ana Karen Ram\'irez L\'opez<br>
75          </h5>
76        </div> <!--FIN DEL FOOTER-->
77      </div><!--FIN DEL CONTAINER-->
78    </div><!--FIN DEL SHADOW-->
79 </body>
80 </html>
```

SECCIÓN E.3

publico.html

```
1  <!DOCTYPE html>
2  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.
       org">
3
4  <head>
5  <meta charset="utf-8">
6  <title>Mi sitio web</title>
7  <link href="css/Style_web.css" rel="stylesheet" type="text/css">
8  </head>
9
10 <body>
```

```
<!--INICIO DEL SHADOW-->
    <div id="shadow"></div>
<!--INICIO DEL CONTAINER-->
<div id="container">

      <div id="header">
        <img src="web_img/img_header.jpg" width="960" height="191">
      </div> <!--FIN DEL HEADER-->

      <div id="nav">
      <ul>
     <li><a th:href="@{/}" name="Inicio" id="home.html">Inicio</a></li>
     <li><a th:href="@{/historia}" name="Historia" id="historia.html">
     Historia</a></li>
     <li><a th:href="@{/publico}" name="Publico" id="publico.html">
     Publico</a></li>
     <li><a th:href="@{/privado}" name="Privado" id="privado.html">
     Privado</a></li>
      </ul>
      </div>
      <!--FIN DEL NAV-->

     <div id="article">

     <div id="section">
     <h1>Introducci\'on</h1>
   <h3>
   <p align=justify>
    
   Esta p\'agina est\'a hecha para respaldar el trabajo escrito, donde se
    presenta diferentes tipos de generadores m\'as comunes, adem\'as se
    muestran algunos ejemplos, donde se aplicaron algunas pruebas de
    aleatoriedad para los n\'umeros pseudoaleatorios generados.</p>

     <p align=justify>
    
     Recordemos que los n\'umeros pseudoaleatorios se obtienen a partir de
    un generador, que es el que se ocupa en la siguiente pesta\~na, que
```

```
produce una sucesi\'on de valores que se pretende sean una secuencia de
variables aleatorias independientes y distribuidas uniformemente y que
no se repitan.</p>

  <p align=justify>
 
  Entonces la idea principal para esta p\'agina es que en la pesta\~na
privada sirva para la implementaci\'on del generador, para saber c\'omo
se desarroll\'o este trabajo, revisar el escrito.</p>

  <p align=justify>
 
  S\'olo con la aplicaci\'on m\'ovil, se podr\'a ingresar.</p>
  </h3>
  </div><!--FIN DEL SECTION-->

  </div><!--FIN DEL ARTICLE-->

   <div id="aside">
   <h3>
   <br><br><br>
   Universidad Aut\'onoma de la Ciudad de M\'exico <br><br>
   San Lorenzo Tezonco <br><br>
   Calle Prolongaci\'on San Isidro No. 151, <br>
   Col. San Lorenzo Tezonco, <br>
   Del. Iztapalapa, M\'exico, CDMX, <br>
   C.P. 09790. <br>
   <br><br><br>
   <h3><a href="https://www.uacm.edu.mx/">Sitio de la UACM</a></h3>
   </h3>
   </div> <!--FIN DEL ASIDE-->

   <div id="footer">
    <h5>
    \copyright2020 <br>
    P\'agina personal <br>
    Ana Karen Ram\'irez L\'opez<br>
    </h5>
```

```
78        </div> <!--FIN DEL FOOTER-->
79      </div><!--FIN DEL CONTAINER-->
80    </div><!--FIN DEL SHADOW-->
81 </body>
82 </html>
```

SECCIÓN E.4

privado.html

```
1  <!DOCTYPE html>
2  <html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.
       org">
3
4  <head>
5  <meta charset="utf-8">
6  <title>Mi sitio web</title>
7  <link href="css/Style_web.css" rel="stylesheet" type="text/css">
8  </head>
9
10 <body>
11 <!--INICIO DEL SHADOW-->
12     <div id="shadow"></div>
13 <!--INICIO DEL CONTAINER-->
14 <div id="container">
15
16       <div id="header">
17         <img src="web_img/img_header.jpg" width="960" height="191">
18       </div> <!--FIN DEL HEADER-->
19
20       <div id="nav">
21       <ul>
22      <li><a th:href="@{/}" name="Inicio" id="home.html">Inicio</a></li>
23      <li><a th:href="@{/historia}" name="Historia" id="historia.html">
24       Historia</a></li>
25      <li><a th:href="@{/publico}" name="Publico" id="publico.html">
26       Publico</a></li>
27      <li><a th:href="@{/privado}" name="Privado" id="privado.html">
```

```
28      Privado</a></li>
29       </ul>
30       </div>
31       <!--FIN DEL NAV-->
32
33      <div id="article">
34      <div class="jumbotron">
35     <h1> Bienvenidos al Token </h1>
36       <h3>
37       En el primer recuadro genera tu semilla, en el cual lo introducir\'as<
    br>
38       en tu aplicaci\'on m\'ovil.<br>
39       Para que despu\'es con ella puedas entrar a la p\'agina.<br><br>
40       </h3>
41
42          <form th:action="@{/}" method="post">
43          <input type="text" placeholder="Semilla" name="semilla" id="semilla"
     th:value="${semilla}" readonly/><br/><br/>
44          <input type="submit" id="calcular" name="calcular" value="calcular"
    class="btn btn-info"/>
45          </form>
46          <br/><br/>
47          <form th:action="@{/valida}" method="post">
48          <input type="text" placeholder="Ingresa tu numero" name="resultado"
    id="resultado" th:value="${resultado}"/><br/><br/>
49          <input type="submit" id="calcular" name="calcular" value="calcular"
    class="btn btn-info"/>
50          </form>
51          <br/><br/>
52
53          <span class="label label-warning" th:text="${mensajeFracaso}"> </
    span>
54
55        </div> <!--FIN DEL JUMBOTRON-->
56
57        </div> <!--FIN DEL ARTICLE-->
58
59       <div id="aside">
```

```
60        <h3>
61        <br><br><br>
62        Universidad Aut\'onoma de la Ciudad de M\'exico <br><br>
63        San Lorenzo Tezonco <br><br>
64        Calle Prolongaci\'on San Isidro No. 151, <br>
65        Col. San Lorenzo Tezonco, <br>
66        Del. Iztapalapa, M\'exico, CDMX, <br>
67        C.P. 09790. <br>
68        <br><br><br>
69        <h3><a href="https://www.uacm.edu.mx/">Sitio de la UACM</a></h3>
70        </h3>
71        </div> <!--FIN DEL ASIDE-->
72
73        <div id="footer">
74         <h5>
75         \copyright2020 <br>
76         P\'agina personal <br>
77         Ana Karen Ram\'irez L\'opez<br>
78         </h5>
79        </div> <!--FIN DEL FOOTER-->
80      </div><!--FIN DEL CONTAINER-->
81    </div><!--FIN DEL SHADOW-->
82 </body>
83 </html>
```

SECCIÓN E.5

pagprivado.html

```
1 <!DOCTYPE html>
2 <html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.
      org">
3
4 <head>
5 <meta charset="utf-8">
6 <title>Mi sitio web</title>
7 <link href="css/Style_web.css" rel="stylesheet" type="text/css">
8 </head>
```

```
9
10 <body>
11 <!--INICIO DEL SHADOW-->
12     <div id="shadow"></div>
13 <!--INICIO DEL CONTAINER-->
14 <div id="container">
15
16       <div id="header">
17         <img src="web_img/img_header.jpg" width="960" height="191">
18       </div> <!--FIN DEL HEADER-->
19
20       <div id="nav">
21       <ul>
22      <li><a th:href="@{/}" name="Inicio" id="home.html">Inicio</a></li>
23      <li><a th:href="@{/historia}" name="Historia" id="historia.html">
24      Historia</a></li>
25      <li><a th:href="@{/publico}" name="Publico" id="publico.html">
26      Publico</a></li>
27      <li><a th:href="@{/privado}" name="Privado" id="privado.html">
28      Privado</a></li>
29       </ul>
30       </div>
31       <!--FIN DEL NAV-->
32
33       <div id="article">
34
35       <div id="section">
36       <h1> Felicidades </h1>
37       <h2>
38       <div align='center'>
39       Acabas de entrar a la p\'agina privada <br>
40       <span class="label label-warning" th:text="${mensajeExito}"></span>
41       <span th:text="${resultado}"></span><br>
42       Aqu\'i tienes el c\'odigo del token con el que se trabaj\'o.<br><br>
43       </div>
44       </h2>
45      <iframe src="https://drive.google.com/file/d/11
    vuMXsymu8Id4gXVQfAER_XmtpHCGF4K/preview" width="640" height="480"></
```

```
    iframe>
46
47          </div><!--FIN DEL SECTION-->
48
49          </div><!--FIN DEL ARTICLE-->
50
51        <div id="aside">
52        <h3>
53        <br><br><br>
54        Universidad Aut\'onoma de la Ciudad de M\'exico <br><br>
55        San Lorenzo Tezonco <br><br>
56        Calle Prolongaci\'on San Isidro No. 151, <br>
57        Col. San Lorenzo Tezonco, <br>
58        Del. Iztapalapa, M\'exico, CDMX, <br>
59        C.P. 09790. <br>
60        <br><br><br>
61        <h3><a href="https://www.uacm.edu.mx/">Sitio de la UACM</a></h3>
62        </h3>
63        </div> <!--FIN DEL ASIDE-->
64
65        <div id="footer">
66          <h5>
67          \copyright2020 <br>
68          P\'agina personal <br>
69          Ana Karen Ram\'irez L\'opez<br>
70          </h5>
71        </div> <!--FIN DEL FOOTER-->
72      </div><!--FIN DEL CONTAINER-->
73    </div><!--FIN DEL SHADOW-->
74 </body>
75 </html>
```

BIBLIOGRAFÍA

[1] Guerrero, C. *Unidad II. Números aleatorios y pseudo aleatorios.* Consultado Agosto 29, 2019, de Academia edu Sitio web: `https://independent.academia.edu/CHRISTIANGUERRERO4`

[2] Mancilla, A. *Números aleatorios. Historia, teoría y aplicaciones.* Consultado Agosto 31, 2019, de Sistema de Información Científica Redalyc Red de Revistas Científicas de América Latina y el Caribe, España y Portugal Sitio web: `http://www.redalyc.org/articulo.oa?id=85200804`

[3] Palacio, C. *Método de Montecarlo.* Consultado Enero 8, 2019, de Universidad Autónoma de Madrid Sitio web: $https://web.archive.org/web/20190108165041/http://www.uam.es:80/personal_pdi/ciencias/carlosp/html/pid/montecarlo.html$

[4] Coss, R. *Simulación un enfoque práctico.* México: Limusa.

[5] Jay L. Devore. (2008). *Probabilidad y Estadística para Ingeniería y Ciencias.* México: Cengage Learning.

[6] García E., García H., Cárdenas L. (2013). *Simulación y análisis de sistemas con ProModel.* México: Pearson.

[7] Yagüe, C. (2019). *Qué es Apache Maven*. Consultado Febrero 26, 2020, de openwebinars Sitio web: https://openwebinars.net/blog/que-es-apache-maven/

[8] (2013). *PHP: WampServer Defición, istalación y configuración*. Consultado Febrero 26, 2020, Sitio web: https://codegeando.blogspot.com/2013/03/php-wampserver-definicion-instalacion-y.html

[9] Ayala J. (2012). *Entorno de desarrollo Java – Spring Tool Suite*. Consultado Febrero 26, 2020, Sitio web: http://jmaw.blogspot.com/2012/09/entorno-de-desarrollo-java-spring-tool.html

[10] Miranda M. (2018). *Estadística computacional*. Consultado Enero 10, 2021, Sitio web: https://rpubs.com/mauriciom/383885

[11] Martinez I. (2018). *Regla de Sturges*. Consultado Enero 10, 2021, Sitio web: http://imarranz.com/sturges.html

[12] Machado S., Mendes R., Pereira E., Jorge S., Derci A. (2010). *DISTRIBUTION OF TOTAL HEIGHT, TRANSVERSE AREA AND INDIVIDUAL MACHADO, S. do A. et al. VOLUME FOR Araucaria angustifolia (Bert.) O. Kuntze*. Brasil Consultado Enero 10, 2021, Sitio web: https://www.redalyc.org/pdf/744/74415015002.pdf